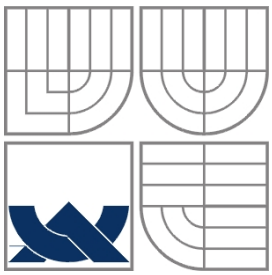
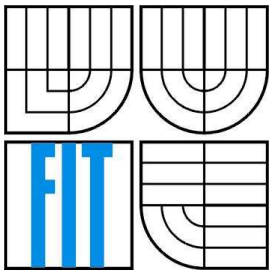


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

KOREKTOR DIAKRITIKY

AUTOMATIC GENERATOR OF DIACRITICS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

LUKÁŠ VESELÝ

VEDOUCÍ PRÁCE
SUPERVISOR

Doc. RNDr. PAVEL SMRŽ, Ph.D.

BRNO 2007

Zadání diplomové práce/6149/2006/xvesel23

Vysoké učení technické v Brně - Fakulta informačních technologií

Ústav počítačové grafiky a multimédií Akademický rok 2006/2007

Zadání diplomové práce

Řešitel: **Veselý Lukáš**
Obor: Inteligentní systémy
Téma: **Korektor diakritiky**
Kategorie: Umělá inteligence

Pokyny:

1. Seznamte se s datovými strukturami a algoritmy pro efektivní uložení slovníku.
2. Na základě existující implementace systému pro minimalizaci konečného automatu a procházení strukturou trie navrhněte a realizujte systém, který doplní do českého textu diakritiku (háčky a čárky).
3. Vytvořte multiplatformní prostředí pro interaktivní práci s vytvořeným systémem.
4. Vyhodnoťte implementovaný systém z hlediska časové náročnosti doplňování diakritiky.

Literatura:

- Jan Daciuk, Stoyan Mihov, Bruce W. Watson, Richard Watson: Incremental Construction of Minimal Acyclic Finite State Automata. Computational Linguistics 26(1): 3-16 (2000)

Při obhajobě semestrální části diplomového projektu je požadováno:

1. Znalost příslušných struktur a algoritmů.
2. Prototyp systému.

Podrobné závazné pokyny pro vypracování diplomové práce naleznete na adrese http://www.fit.vutbr.cz/info/szz/

Technická zpráva diplomové práce musí obsahovat formulaci cíle, charakteristiku současného stavu, teoretická a odborná východiska řešených problémů a specifikaci etap, které byly vyřešeny v rámci ročníkového a semestrálního projektu (30 až 40% celkového rozsahu technické zprávy).

Student odevzdá v jednom výtisku technickou zprávu a v elektronické podobě zdrojový text technické zprávy, úplnou programovou dokumentaci a zdrojové texty programů. Informace v elektronické podobě budou uloženy na standardním paměťovém médiu (disketa, CD-ROM), které bude vloženo do písemné zprávy tak, aby nemohlo dojít k jeho ztrátě při běžné manipulaci.

Vedoucí: **Smrž Pavel, doc. RNDr., Ph.D.**, UPGM FIT VUT
Datum zadání: 28. února 2007
Datum odevzdání: 22. května 2007

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta informačních technologií
Ústav počítačové grafiky a multimédií
612 66 Brno, Božetěchova 2

L.S.

doc. Dr. Ing. Pavel Zemčík
vedoucí ústavu

LICENČNÍ SMLOUVA
POSKYTOVANÁ K VÝKONU PRÁVA UŽÍT ŠKOLNÍ DÍLO

uzavřená mezi smluvními stranami

1. Pan

Jméno a příjmení: **Lukáš Veselý**
Id studenta: 18521
Bytem: Hochmanova 11, 628 00 Brno
Narozen: 05. 10. 1981, Hodonín
(dále jen "autor")

a

2. Vysoké učení technické v Brně

Fakulta informačních technologií
se sídlem Božetěchova 2/1, 612 66 Brno, IČO 00216305
jejímž jménem jedná na základě písemného pověření děkanem fakulty:

.....
(dále jen "nabyvatel")

Článek 1
Specifikace školního díla

1. Předmětem této smlouvy je vysokoškolská kvalifikační práce (VŠKP):
diplomová práce

Název VŠKP: Korektor diakritiky
Vedoucí/školitel VŠKP: Smrž Pavel, doc. RNDr., Ph.D.
Ústav: Ústav počítačové grafiky a multimédií
Datum obhajoby VŠKP: .....

VŠKP odevzdal autor nabyvateli v:
tištěné formě počet exemplářů: 1
elektronické formě počet exemplářů: 2 (1 ve skladu dokumentů, 1 na CD)

2. Autor prohlašuje, že vytvořil samostatnou vlastní tvůrčí činností dílo shora popsané a specifikované. Autor dále prohlašuje, že při zpracovávání díla se sám nedostal do rozporu s autorským zákonem a předpisy souvisejícími a že je dílo dílem původním.
3. Dílo je chráněno jako dílo dle autorského zákona v platném znění.
4. Autor potvrzuje, že listinná a elektronická verze díla je identická.

Článek 2
Udělení licenčního oprávnění

1. Autor touto smlouvou poskytuje nabyvateli oprávnění (licenci) k výkonu práva uvedené dílo nevýdělečně užít, archivovat a zpřístupnit ke studijním, výukovým a výzkumným účelům včetně pořizování výpisů, opisů a rozmnoženin.
2. Licence je poskytována celosvětově, pro celou dobu trvání autorských a majetkových práv k dílu.
3. Autor souhlasí se zveřejněním díla v databázi přístupné v mezinárodní síti:
   - ☐ ihned po uzavření této smlouvy
   - ☒ 1 rok po uzavření této smlouvy
   - ☐ 3 roky po uzavření této smlouvy
   - ☐ 5 let po uzavření této smlouvy
   - ☐ 10 let po uzavření této smlouvy

   (z důvodu utajení v něm obsažených informací)
4. Nevýdělečné zveřejňování díla nabyvatelem v souladu s ustanovením § 47b zákona č. 111/ 1998 Sb., v platném znění, nevyžaduje licenci a nabyvatel je k němu povinen a oprávněn ze zákona.

Článek 3
Závěrečná ustanovení

1. Smlouva je sepsána ve třech vyhotoveních s platností originálu, přičemž po jednom vyhotovení obdrží autor a nabyvatel, další vyhotovení je vloženo do VŠKP.
2. Vztahy mezi smluvními stranami vzniklé a neupravené touto smlouvou se řídí autorským zákonem, občanským zákoníkem, vysokoškolským zákonem, zákonem o archivnictví, v platném znění a popř. dalšími právními předpisy.
3. Licenční smlouva byla uzavřena na základě svobodné a pravé vůle smluvních stran, s plným porozuměním jejímu textu i důsledkům, nikoliv v tísni a za nápadně nevýhodných podmínek.
4. Licenční smlouva nabývá platnosti a účinnosti dnem jejího podpisu oběma smluvními stranami.

V Brně dne: .....

.....	.....
Nabyvatel	Autor

Abstrakt

Cílem předkládané práce je návrh a implementace aplikace, umožňující doplňování a naopak odstranění diakritiky v česky psaném textu. Popsána je použitá vyhledávací struktura trie a její vztah s konečnými automaty. Dále je předveden algoritmus minimalizace konečného automatu a diskutovány různé metody pro doplňování diakritiky. V praktické části je uvedena samotná implementace programu v programovacím jazyce Java s využitím objektově orientovaného přístupu. Na závěr je provedeno vyhodnocení a analýza dosažených výsledků.

Klíčová slova

diakritika, Java, konečný automat, minimalizace, slovník, struktura trie, unigramový model

Abstract

The goal of this diploma work is the suggestion and the implementation of the application, which allows adding / removing of diacritics into / from Czech written text. Retrieval “trie” structure is described along with its relation to finite state automata. Further, algorithm for minimization of finite state automata is described and various methods for adding diacritics are discussed. In practical part the implementation in Java programming language with usage of object-oriented approach is given. Achieved results are evaluated and analysed in the conclusion.

Keywords

diacritic, Java, finite state automata, minimalization, dictionary, structure trie, unigram model

Citace

Veselý Lukáš: Korektor diakritiky. Brno, 2007, diplomová práce, FIT VUT v Brně.

Korektor diakritiky

Prohlášení

Prohlašuji, že jsem diplomový projekt vypracoval samostatně pod vedením RNDr. Pavla Smrže, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Jméno Příjmení
Datum

Poděkování

Děkuji RNDr. Pavlovi Smržovi, Ph.D. za odborné vedení, za trpělivost a za čas strávený nad mým projektem.

Obsah

1 Úvod

V současné době je téměř každý osobní počítač vybaven softwarem, který umožňuje zpracovávat texty. Mezi nejrozšířenější patří různé textové editory nebo systémy pro počítačovou sazbu. Tyto programy obvykle uživateli poskytují funkce umožňující řešit běžné problémy, které při psaní textů vznikají.

Vedle standardních operací týkajících se vnější úpravy textu, kam patří mazání a vkládání částí textu, grafická úprava a podobně, existuje celá řada problémů spjatých s jazykovou stránkou takto vznikajících textů. I zde je snahou opakující se procesy co nejvíce automatizovat. Hovoří se o tzv. jazykové podpoře programů. Do oblasti jazykové podpory spadají programy pro automatické dělení slov na konci řádku, automatické korektory pravopisu, počítačové *tezaury* (synonymické slovníky nabízející v interaktivním režimu uživateli výběr vhodného jazykového výrazu), programy pro automatickou gramatickou korekturu textu, pro stylistické úpravy atd. Programy tohoto druhu mohou, a ve většině případů také, alespoň z části, využívají morfologickou analýzu.

Cílem této práce je vytvořit a implementovat jazykovou podporu v podobě obnovení diakritiky (háčků a čárek) v českém textu. Je nanejvýš žádoucí, aby automatizace tohoto procesu byla co možná nejvyšší a současně vykazovala malé procento chybovosti.

Na úvod se zaměříme na důvody, které nás vedly k rozhodnutí zabývat se touto problematikou. Podíváme se, proč je vhodné psát s diakritikou nebo naopak bez ní a do jaké míry bude tato práce prospěšná.

Hlavním faktorem ovlivňujícím efektivitu algoritmu pro obnovu diakritiky je vhodné uložení a rychlé vyhledávání výrazů ve slovníku. Ve třetí kapitole si ukážeme, proč je pro tento účel nejvhodnější výběr vyhledávací struktury trie, budou naznačeny její vlastnosti a způsoby uložení v paměti počítače.

Nevýhodou trie je značná prostorová složitost. Při snaze o její snížení využijeme analogie mezi strukturou trie a deterministickým konečným automatem. Jako vhodný algoritmus minimalizace počtu stavů konečného automatu si přiblížíme algoritmus Jana Daciuka, který bude poté aplikován na samotnou strukturu trie. Na formálním popisu vztahu mezi deterministickým konečným automatem a trie si ukážeme jejich úzkou spojitost.

Ve čtvrté kapitole se zaměříme na různé metody, které se zabývají doplňováním diakritiky. Nastíníme některé z nich a ukážeme si, jestli je výhodnější zpracovávat jednotlivá slova po znacích, soustředit se na jejich syntaktickou analýzu, nebo vycházet z četnosti jejich výskytu v datovém korpusu.

Pro korektní fungování celé aplikace je důležité vhodné vytvoření slovníku českých slov, ze kterého budeme při následném doplňování diakritiky vycházet. Možností je několik a hledání optimální varianty bude předmětem páté kapitoly. Budou zde prezentovány výsledky, které byly

získány na základě analýzy testovacích dat. Tato data představují část uceleného korpusu, který je vhodný pro získávání informací využitých při sestavování slovníku. Kapitola vychází ze semestrálního projektu, na který tato diplomová práce navazuje.

V šesté kapitole se pak detailně seznámíme se samotnou aplikací pro doplňování diakritiky. Nastíníme specifikaci programu a budeme se zabývat vybranými konkrétními implementačními postupy - vysvětlíme samotný princip doplňování diakritiky, dotkneme se odstraňování překlepů ve slovníku, stanovíme hranici pro automatický překlad atd.

Na závěr budeme diskutovat úspěšnost aplikace z pohledu uživatele. Porovnáme rychlost a kvalitu překladu pro různé stupně automatizace a zmíníme možnosti praktického využití pro běžné uživatele.

2 Motivace doplňování diakritiky

V mnoha středoevropských jazycích tvoří diakritika nedílnou součást spisovného jazyka. Diakritický pravopis u nás nahradil pravopis spřežkový v 15. století a za jeho zakladatele je obecně považován Mistr Jan Hus. Autorství je sice v některých sférách diskutováno, ale jeho doporučení a návrhy zveřejněné v díle De ortographia Bohemica (O pravopise českém) měla každopádně dalekosáhlé následky. Je zjevné, že tyto změny vnímal jako výrazné zjednodušení českého pravopisu, ale v dnešní době se mu od příslušníků počítačové generace dostává spíše kritika.

2.1 Význam diakritiky

Čeština patří k jazykům, ve kterých se setkáváme s diakritikou velmi často. Připomeňme, že se v ní vyskytuje celkem 15 písmen, které mají diakritiku, samozřejmě i s ekvivalenty velkých písmen a tvoří tak třináct různých samostatných skupin (do jedné skupiny řadíme *é*, *ě* a také *ú*, *ů*).

Naproti tomu v angličtině se diakritika nevyskytuje vůbec. Pokud bereme v úvahu všechny jazyky, ve kterých se píše latinkou, je angličtina jediný jazyk bez diakritiky. Existuje v ní sice několik slov, která diakritiku obsahují (fiancé, café), ale tato slova jsou přejatá z cizích jazyků (většinou z francouzštiny), a navíc k sobě nemají odpovídající slovo bez diakritiky, se kterým by bylo zaměnitelné. Hůře než čeština je na tom ještě slovenština (16 písmen s diakritikou) a holandština (21 písmen). Je tedy zřejmé, že ve většině jazyků je automatická korekce velkým přínosem.

Na druhé straně se ale nabízí otázka, do jaké míry je v soudobém světě diakritika důležitá. Proč se s ní vůbec trápit, když průměrně vzdělaný mluvčí daného jazyka rozumí textu bez háčků a čárek prakticky stejně dobře, jako s nimi.

Když nahlédneme do slovníku cizích slov, termín *diakritický* je definován jako *rozlišovací, rozeznávací*. A právě k rozlišení různého významu slouží diakritická znaménka, tedy háčky, čárky, kroužky a podobně. Jejich užívání nenajdeme samozřejmě jen v českém jazyce, naopak, je jen velmi málo jazyků, v nichž se diakritická znaménka vůbec nevyskytují (například angličtina).

Uvedeme jen nepatrný zlomek případů, kdy diakritika hraje významnou roli při rozlišování významu jednotlivých slov:

- cela – čela
- rada – řada
- citelný – čitelný
- jed – jeď
- sraz – sráz
- jez – jež
- plast – plášť

Je zcela zřejmé, že diakritika má v českém pravopise velmi důležitou úlohu. Problémem může být spíše to, zda je opravdu tak nutné ji v písemné komunikaci využívat a pokud ano, tak při jaké příležitosti. Už bylo zmíněno, že si uživatelé počítače v neformálním projevu psaním háčků a čárek moc hlavu nelámou, což může mít několik důvodů.

Jednou z příčin je například zvyk z dob, kdy nebyla čeština na internetu tak velmi rozšířena, kdy používané kódování nebylo na takové úrovni a českým znakům s diakritickými znaménky příliš nevycházelo vstříc (posílání e-mailových zpráv).

Další příčinou může být automatizace nabytá při programování nebo při stále oblíbenějším psaní textových zpráv. Tato příčina je poměrně pochopitelná a zvlášť u profesionálních programátorů nebo velmi aktivních pisatelů SMS zpráv může být náročné se přeorientovat na uvažování s háčky a čárkami.

Velmi rozšířeným důvodem je větší či menší míra pohodlnosti. Pro řadu uživatelů, kteří neumí psát plynule s použitím horního patra klávesnice, je snazší a rychlejší sdělit informace *cesky*. Programátorům může být navíc proti mysli přepínat klávesnici ze svého obvyklého rozložení na českou variantu.

Pro příležitostné ignorování diakritiky tedy existuje celá řada zdůvodnění, z nichž přinejmenším některá mají poměrně racionální základ (problémy s kódováním při psaní e-mailů), nebo jsou zcela nenapadnutelná (příspěvky psané z ciziny).

Přes všechny argumenty pro nepoužívání diakritiky existuje samozřejmě názor, který se diakritiky zastává. Nalezneme tedy pro používání diakritiky nějaké rozumné důvody? Určitě ano, budeme se snažit některé z nich uvést.

a) Slušnost - Obecně se dá říci, že čas, který autor jednorázově ušetří při psaní, ztrácejí opakovaně všichni jeho čtenáři při snaze dešifrovat jeho sdělení. Používání diakritiky, zvláště jedná-li se o text delší, je tedy projevem určité ohleduplnosti vůči čtenářům.
b) Čitelnost - Není pochyb, že text s diakritikou se snad každému čte lépe, než text bez ní. Navíc nepoužívání háčků a čárek je často provázeno také ignorací dalších doporučení, jako je například logické dělení delšího textu do odstavců, používání podnadpisů a podobně. Text se pak stává daleko méně přehledný.
c) Jazyková správnost - I v této oblasti platí, že opomíjení diakritiky s sebou přináší i další problémy. Poměrně častou součástí *cestiny* je i absence interpunkce, ignorování velkých písmen (i na začátku vět) a podobně. Když se následně přidá nějaký ten překlep nebo záměna, získáme text, který je v mnoha případech velmi nesrozumitelný.

Používat vůbec diakritiku? Dát na tuto otázku všeobecně platnou odpověď je ve své podstatě nemožné. Vždy záleží na okolnostech a v neposlední řadě i na technických možnostech. Například používání diakritiky v komunikaci přes e-mail není dnes ve většině případů spojeno s žádnými

závažnými technickými obtížemi (pomineme-li spojení se zahraničím). Zvláště v případě, kdy se jedná o formální komunikaci, by tedy mělo být používání diakritiky normou.

Na druhé straně stojí různé neformální diskuse, kdy při stručném příspěvku diakritika zapotřebí být nemusí, ale pokud chceme sdělit něco zásadnějšího, dá se o jejím použití minimálně uvažovat.

Samostatnou kapitolu tvoří také výrazy a slovní spojení, které používáme pro vyhledávání informací v různých vyhledávačích (Seznam, Google atd.). Nabízí se otázka, jestli Češi při vyhledávání zadávají dotazy s háčky a čárkami nebo bez nich, případně v jakém poměru. Podle dostupného testu na [JAN-04] pomocí AdWords na vyhledávači Google, byly zjištěny více či méně překvapivé výsledky. Usuzuje se, že lidé hledají s diakritikou průměrně pětkrát častěji než bez diakritiky. Z toho vyplývá, že ne všichni uživatelé na správnou češtinu zanevřeli.

Ať chceme nebo ne, pokud vezmeme do ruky tužku a papír, jsou pro nás háčky a čárky nutnou samozřejmostí. A pokud používáme klávesnici, tak by ve většině případů diakritika měla také tvořit nedílnou součást našeho jazykového projevu.

Ukázali jsme si, že používání diakritických znamének v běžném projevu je vhodné, ve formální korespondenci přímo nutné, ale ne vždy pohodlné, rychlé a efektivní. A právě využití aplikace, která je součástí této diplomové práce, by mělo tento problém řešit. Stále častěji je potřeba konkrétní texty doplňovat diakritikou nebo ji z nich naopak odstraňovat. Pokud se jedná o slovo nebo odstavec, není to až takový problém, jako kdybychom měli opravovat rozsáhlé texty. Aby samotná aplikace byla přínosem, měla by být práce s ní efektivnější než korekce oprav uživatelem standardním postupem. To se budeme snažit zhodnotit na závěr v celkovém shrnutí.

2.2 Analýza doplňování diakritiky

V ideálním případě by výsledkem práce měl být systém, které převede text se stoprocentní správností. Pro mnoho slovních tvarů však bohužel není doplnění diakritiky jednoznačné. To znamená, že diakritiku můžeme přidat na různá místa ve slově a dostaneme rozdílná slova. Nastávají případy, kdy se změnou háčku nebo čárky změní pouze tvar slova:

- ojediněle - ojedinělé
- dni – dní

Jindy dojde ke změně celého významu:

- jedna – jedná
- cele – celé – čele

Aby bylo zřejmé, který tvar je při doplňování správný, je možné vycházet ze zbylého textu, který je potřeba analyzovat, a na základě kontextu rozhodnout, jaký tvar z nabízených možností je nejpravděpodobnější. Vzhledem ke snaze o minimalizaci paměťových nároků a k vysoké míře

úspěšnosti metody je však pro interaktivní aplikaci možno použít i podstatně jednodušší způsob, který vždy pro daný tvar bez diakritiky seřadí možné tvary s doplněnou diakritikou podle četnosti jejich výskytu nezávisle na kontextu a nechá na uživateli, aby rozhodl, který tvar je v daném kontextu nejpravděpodobnější.

Touto cestou se také ubírá předkládaná diplomová práce. Pravděpodobnost přepisu je určována z velkého množství textových dat (textového korpusu). Při testování bylo zjištěno, že správnost překladu při pouhém nahrazení slova bez diakritiky nejpravděpodobnějším výrazem s diakritikou se pohybuje nad 96%. Tyto výsledky jsou analyzovány a dále diskutovány později v páté kapitole. Následně jsou realizována zlepšení, která pravděpodobnost zvýší za cenu menší automatizace. V samotné aplikaci si uživatel může zvolit poměr mezi automatizací procesu doplňování a požadovanou přesností.

Jestliže chceme vybírat různé varianty překladu, musíme mít tyto slovní tvary vhodně uloženy. Samozřejmě se musí brát v úvahu přístupová doba vyhledávání a minimální paměťové nároky, které jsou důležité pro snadnou přenositelnost aplikace.

Samotné uložení slov do slovníku ovšem nestačí. Abychom dokázali určit četnost výskytu, musíme ke každému z nich navíc přiřadit informaci ohledně pravděpodobnosti přepisu ze slova bez diakritiky. Následující jednoduché příklady to ukazují:

- výraz *cele* se v dostupných textových datech v 71,8% přepíše na *celé*, v 25,9% na *čele* a v 2,3% na *cele;*
- výraz c*tvrtek* se ze 100% přepíše na *čtvrtek*, jiná možnost není.

V následující kapitole se budeme najít vhodnou strukturu, do které bychom mohli náš slovník uložit a následně v ní jednoduše vyhledávat.

3 Efektivní uložení slovníku

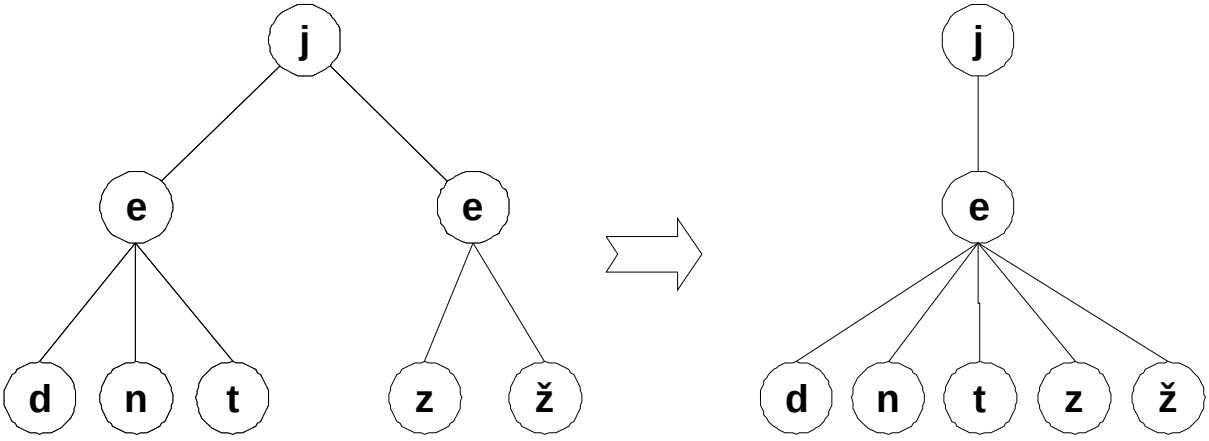
V této části bude představena a diskutována vyhledávací struktura pro efektivní uložení slovní zásoby. Takto uložená data jsou pak nejčastěji využívána v aplikacích, jakými jsou například korektory pravopisu, nástroje pro zpracování přirozeného jazyka a dalších. Typicky jsou to takové systémy, ve kterých je jedním z úkolů test na přítomnost hledaného výrazu ve slovníku. Pokud nedochází v použitých datových strukturách k průběžným změnám obsahu, jsou nazývány statické. Takto vytvořený statický slovník využijeme i v naší práci.

Základní požadavky na datovou strukturu pro uchování slovní zásoby jsou následující:

- **Paměťové nároky** – slovník obsahuje velké množství výrazů a jeho velikost se tak může pohybovat v řádech desítek až stovek megabajtů. Práce s takto velkým objemem dat v paměti by znamenala nadměrnou paměťovou režii a vysoké nároky na výpočetní sílu. Proto je žádoucí objem dat redukovat s pomocí vhodných algoritmických postupů. Využitá struktura by měla být nezávislá na velikosti uložených dat.
- **Rychlost prohledávání** – důležitou vlastností použité vyhledávací struktury je rychlost a způsob jejího procházení. Často je například potřeba přistupovat k velkému množství slov, což u nevhodně zvolené reprezentace může způsobit prodlevy a zpomalení programu.

3.1 Struktura trie

Při bližším nahlédnutí na data, která mají být ve slovnících uložena, si můžeme všimnout pravidelností plynoucích z charakteru přirozeného jazyka. Například v českém slovníku lze díky skloňování nebo časování nalézt mnoho tvarů, jež sdílejí společný začátek. Bylo by tedy výhodné, společné začátky slovních tvarů vhodným způsobem ztotožnit a ukládat pouze jednou. Obrázek 3.1 takovou úpravu demonstruje.



Obrázek 3.1 – Komprese společného prefixu

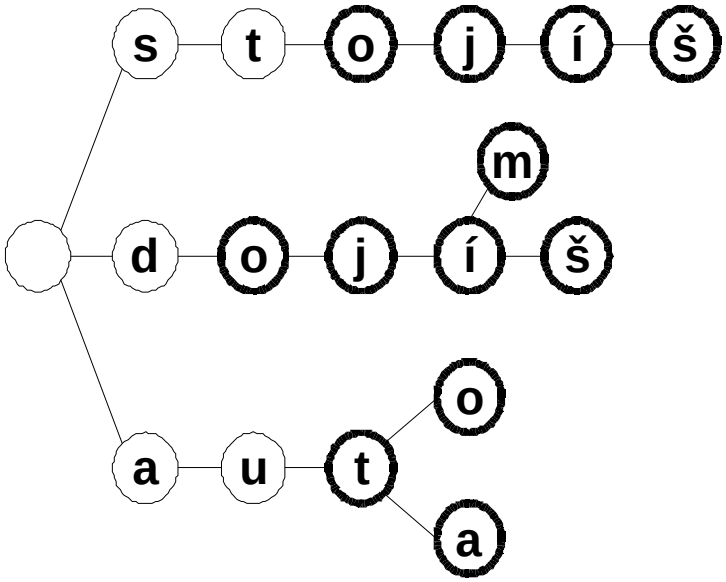
3.1.1 Formální definice struktury *trie*:

Místo toho, abychom založili vyhledávací metodu na porovnávání mezi klíči, využijeme reprezentace jako posloupnosti písmen. Zmíněná struktura se nazývá *trie* a poprvé byla uveřejněna E. Fredkinem v roce 1960, který její pojmenování vysvětluje jako část sousloví *information re**trie**val*.

Definice struktury trie uvedena v [SED99]

Trie je *m*-ární strom, jehož uzly jsou *m*-ární vektory s komponentami odpovídajícími vzájemně různým písmenům. Každý uzel na úrovni *i* reprezentuje množinu všech klíčů, které začínají jistou posloupností *i* písmen, danou odpovídající větví z kořene do daného uzlu. Uzel tak určuje *m*-ární větvení závisející na (*i*+1) ním písmeni.

Nechť M je tedy množina všech slov, která bude takto reprezentována. Klíč je prvek (slovo) této množiny a $\sum$ označuje abecedu. Klíč se chápe jako posloupnost prvků z množiny $\sum$. V trie pak každá cesta z kořene do listu odpovídá právě jednomu klíči reprezentované množiny M a výsledný strom je *n*-ární, kde *n* je rovno $|\sum|$. Příklad struktury trie je zachycen na obrázku 3.2.



Obrázek 3.2 – Struktura trie

Z obrázku je patrné, že pokud dvě slova sdílejí společný prefix, je tento prefix uložen pouze jednou a slova poté sdílejí část společné cesty v grafu. Koncové uzly (na obr. 3.2 zobrazeny tučně) jsou nazývány terminální. V trie jsou pak všechny listy terminální.

3.1.2 Vyhledávání klíče

Vyhledávání probíhá přímočaře. Nechť T je trie, U je ukazatel do struktury a I pozice znaku v klíči K:

1. U := Kořen (T) Nastav ukazatel na kořen trie.
2. I := 1 Nastav I na pozici prvního znaku v klíči.
3. Není-li v seznamu hran vycházejících z U hrana označená K[I], hledání je neúspěšné – skonči.
4. Jinak proveď průchod po této hraně do uzlu U´ a zvyš I o jedna.
5. Je-li I≤|K|, pokračuj bodem 3 pro U= U´.
6. Pokud je U´ koncový, hledání končí úspěšně, jinak neúspěšně.

Pokud chceme zjistit, zda se hledané slovo nachází ve struktuře, začínáme vyhledávat v kořeni. V každém uzlu hledáme jeho syna, který odpovídá dalšímu písmeni hledaného slova. Jednoduše toto hledání probíhá od nejstaršího po nejmladšího syna. Pokud existuje takový syn, najdeme shodu písmene v něm uloženého s patřičným písmenem hledaného slova, hledání syna zastavíme a pokračujeme přečtením dalšího písmene slova a hledáním shody stejným způsobem. V opačném případě končí hledání neúspěšně.

Zvolení vyhledávací struktury trie pro uložení slov ve slovníku je vhodné díky kompresi společných prefixů a také díky velmi rychlému získání informace, že se hledaný prvek ve struktuře nenalézá. Problémem zůstává paměťová náročnost trie. Tu se budeme snažit snížit na minimum pomocí analogie minimalizace konečného automatu.

3.2 Konečný automat

Zopakujeme si stručně základní informace o konečných automatech. Je to teoretický výpočetní model, který se skládá z několika stavů. Mezi těmito stavy přechází na základě symbolů, které čte ze vstupu.

Formálně je podle [KON-06] konečný automat definován jako uspořádaná pětice $(S, \Sigma, \sigma, s, A)$, kde:

- S je konečná množina *stavů*.
- Σ je konečná množina vstupních symbolů, nazývaná *abeceda*.
- σ je tzv. *přechodová funkce* (též *přechodová tabulka*), popisující pravidla přechodů mezi stavy. Může mít buď podobu $S \times A \rightarrow S$ (deterministický automat), nebo $S \times \{A \cup \varepsilon\} \rightarrow P(S)$ (nedeterministický automat).
- s je *počáteční stav*, $s \in S$.
- A je množina *koncových stavů*, $A \subseteq S$.

Konečný automat nemá žádnou další paměť kromě informace o aktuálním stavu a množina jeho stavů je konečná. Na počátku se nachází v definovaném počátečním stavu. Dále v každém kroku přečte jeden symbol ze vstupu a přejde do stavu, který je dán hodnotou. Tato hodnota v přechodové tabulce odpovídá aktuálnímu stavu a přečtenému symbolu. Poté pokračuje čtením dalšího symbolu ze vstupu, dalším přechodem podle přechodové tabulky atd.

Podle přechodové funkce lze dále definovat, jestli v každém bodě tabulky je pouze jeden cílový stav nebo celá množina stavů. Takový automat se nazývá buď deterministický (jeden stav) nebo nedeterministický (více cílových stavů).

3.2.1 Minimalizace konečného automatu

Struktura trie dokáže efektivně reprezentovat společné prefixy klíčů. Pro uložení rozsáhlých slovníků, zejména v případě morfologicky bohatých jazyků, jsou však paměťové nároky stále příliš vysoké. Snažíme se tedy najít způsob, jak dále snížit počet uzlů nutných k reprezentaci všech uložených slovních tvarů. Pro to je velmi výhodné interpretovat vyhledávání v trie jako průchod konečným automatem – analogicky se z vrcholů stávají stavy a z hran přechody. Počáteční uzel potom odpovídá počátečnímu stavu automatu. Pro zmenšení počtu uzlů, které zachovává ekvivalenci generovaných jazyků, použijeme minimalizační algoritmy. Minimalizace konečného automatu se skládá ze dvou částí: eliminace nedosažitelných stavů a sjednocení ekvivalentních stavů.

Při minimalizačním procesu jde především o to, aby výsledný konečný automat přijímal právě stejný jazyk, jako automat původní. Nedosažitelné stavy lze eliminovat prostým procházením přechodového grafu automatu do hloubky. Pro náš případ ani nemá smysl o nedosažitelných stavech uvažovat vzhledem ke způsobu, jakým budeme konečný automat konstruovat. Metod pro minimalizaci je několik, např. přímá aplikace Myhill-Nerodovy věty, která spočívá v postupném tvoření rozkladů. Hlavní součástí minimalizace je klasifikace jednotlivých stavů. Stavy jsou rozděleny do tříd, které reprezentují daný minimální konečný automat. Při prvním rozkladu R_0 jsou stavy děleny na koncové a ostatní (R_0={A, S-A}). Dále jsou tvořeny nové třídy na základě relací již existujících tříd a algoritmus končí pokud následujícím rozkladem již nevzniknou žádné další třídy. Následně se třídy stávají stavy nově vytvořeného minimálního automatu a jsou vytvořeny i odpovídající přechody.

My si podrobněji nastíníme použitý minimalizační algoritmus vytvořený Janem Daciukem a zveřejněný v [DAC2-98], který prování minimalizaci ihned při přidávání nového stavu do automatu. Uvažujme procházení strukturou trie metodou postorder (procházení začíná na levé straně od listů, poté se přejde na pravou stranu a až na konci je přečten kořen) a podívejme se, jak se rozdělení stavů může realizovat. Začínáme s prvním listem. Všechny stavy až k prvnímu větvení (stav s více jak jedním vycházejícím přechodem) musí patřit do různých tříd. Uložíme je do registru stavů, aby byly následně snadno přístupné. Dále už je nemusíme zaměňovat s jinými stavy. Následně začínáme

sledovat další větve, u kterých nejprve určujeme listy. Potřebujeme zjistit, jestli je můžeme zařadit do stejné třídy nebo ne. Nové stavy patří do shodné třídy právě tehdy když:

1. jsou oba stavy koncové nebo není koncový ani jeden
2. oba mají stejný počet výstupních přechodů
3. odpovídající přechody mají stejné ohodnocení
4. odpovídající přechody vedou ke stejným stavům
5. stavy, které jsou následně dosažitelné jsou jedinými reprezentanty jejich tříd

Jestli jsou splněny všechny podmínky, stav je nahrazen staven, nalézajícím se v registru. Pokud podmínky splněny nejsou, stav se stává zástupcem nové třídy a musí být zapsán do registru. Tato procedura vede k minimalizaci automatu.

Při přidávání nových slov do automatu, je potřeba zachovávat jeho minimalitu. Musíme být schopni vykonat současně dva procesy. Vložit nové slovo do automatu a provést jeho minimalizaci. Je potřeba určit, které stavy podléhají změnám při vložení nových slov a zajistit, aby minimalizace proběhla během vkládání slov. K tomu jsou potřeba lexikograficky setříděná data. Pokud totiž přidáme další slovo v pořadí, změní se pouze stavy, kterými musíme projít pro akceptování posledního přidaného slova do automatu. Mění se díky odstraňování stavů nebo přidáváním nových přechodů. Zbytek automatu zůstává nezměněný. Za společný prefix je považován nejdelší prefix v novém slově, který je současně také již existujícím prefixem v automatu. Nové přechody mohou být přidány pouze za poslední stav, který se součástí společného prefixu nově přidávaného slova. Jestliže jsou slova setříděná, společný prefix může obsahovat pouze ty stavy, které representují začátek posledního slova přidaného do automatu (celé slovo, pokud poslední slovo je prefix nového slova). Každé nové slovo může využít buď stavy z existujícího společného prefixu, nebo může přidat nové stavy. Stavy můžou být odstraněny, pouze pokud se nacházejí v cestě, která representuje poslední přidané slovo do automatu (část, která neobsahuje společný prefix). Když je přidáno nové slovo a poslední slovo není prefixem nového slova, potom jsou stavy v této části odstraněny (jsou přepsány jinými identickými stavy) nebo jsou ustanoveny jako nové stavy. Tento postup minimalizace je vyjádřen v minimalizačním algoritmu:

Register := Ø;

while je další slovo na vstupu

Word := další slovo v lexikografickém pořadí;

CommonPrefix := common_prefix(Word);

LastState := δ(q0,CommonPrefix);*

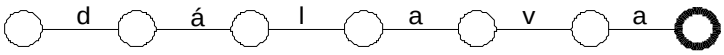
CurrentSuffix := Word[length(CommonPrefix)+1…length(Word)];

if *has_children(LastState)*

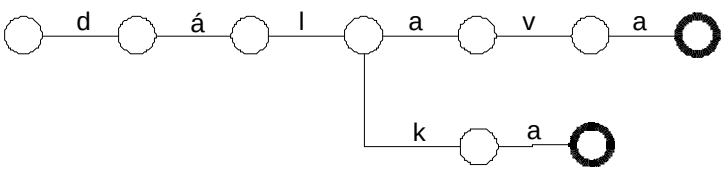
replace_or_register(LastState)

```
        add_suffix(LastState, CurrentSuffix)
replace_or_register(q0)

func common_prefix(Word)
return nejdelší prefix w z Word takový, že δ*(q0,w) ≠ ⊥

func replace_or_register(State)
Child := last_child(State);
if  has_children(Child)
        replace_or_register(Child)
if  last child(State) := q:
        delete(Child)
else Register := Register U {Child}
```

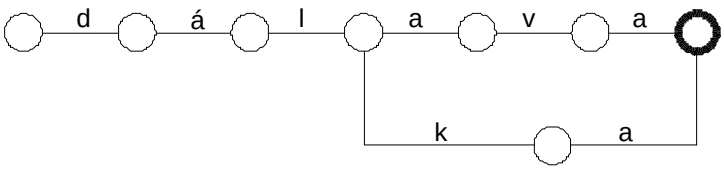
Funkce *common_prefix* vrací nejdelší prefix slova, které je přidáváno a zároveň je shodný s prefixem slova, které se již v automatu nachází. Funkce *add_suffix* tvoří rozšíření automatu, které representuje sufix přidávaného slova (část slova, která zůstala díky společnému prefixu byla vynechána). Poslední stav tohoto rozšíření je označen jako koncový. Funkce *last_child* vrací stav, který je dosažený z počátečního stavu. Jelikož jsou data setříděna, *last_child* vrací stav, který je dosažen přechodem, který byl přidán jako poslední (při přidávání předchozího slova). Každý stav sebou nese informaci, která určuje, jestli již byl nebo nebyl uložen do registru. Je to nutné, jelikož je potřeba určit, které stavy již byly zpracovány. Části automatu jsou ponechány pro následnou úpravu (přepsání nebo uložení do registru), dokud není přidáno další slovo tak, že tyto stavy již dále nepatří to cesty, která přijímá nové slovo. Tato informace o stavu je přečtena pomocí *marked_as_registed* a nastavena v *mark_as_registered*. Nakonec *has_children* vrací *true*, jestliže jsou ze stavu vycházející přechody a *delete_branch* odstraní jeho výchozí stav a všechny stavy, které mohou být z něj dosaženy (jestliže již nejsou označeny jako registrované).



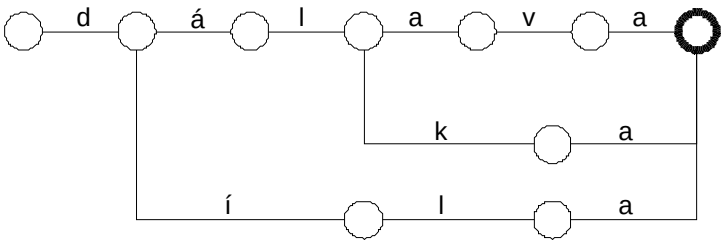
Obrázek 3.3: Konečný automat pro slovo *dálava*



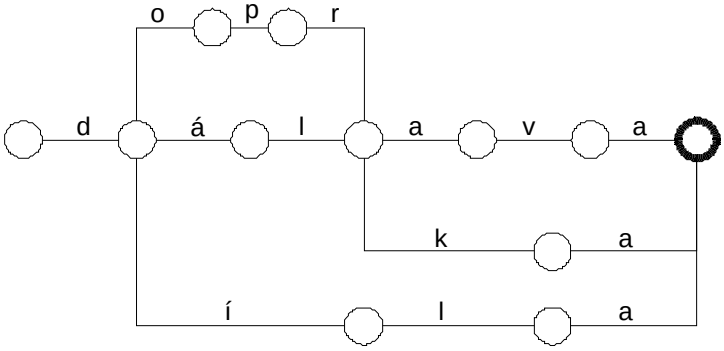
Obrázek 3.4: Byl nalezen společný prefix a vložen zbytek dalšího slova



Obrázek 3.5: Koncový stav slova *dálka* byl nahrazen již existujícím stavem v automatu



Obrázek 3.6: Analogicky provedeno s dalším slovem



Obrázek 3.7: Poslední čtyři stavy pro slovo *doprava* byly využity s existujícího automatu

Požadavky na paměť se stupňují ke konci prováděného výpočtu. Paměť je potřebná zejména pro výstavbu minimalizovaného automatu, volání zásobníku a pro registr stavů. Velikost použité paměti je úměrná počtu stavů a celkovému počtu přechodů. Paměť vyhrazená pro registr stavů je úměrná počtu stavů a může být ihned uvolněna, jakmile je minimalizace dokončena.

Hlavní cyklus algoritmu je prováděn *m*-krát, kde *m* je počet slov, které mají být přijaty automatem. Časová náročnost funkce *common_prefix* je $O(|w|)$, kde $|w|$ je maximální délka slova. Funkce *replace_or_register* je prováděna rekurzivně, nanejvýš $|w|$-krát pro každé slovo. Při každém rekurzivním volání se provede jedno hledání v registru a maximálně jedenkrát se do registru vloží. Maximální časová složitost vyhledávání je $O(log\ n)$, kde *n* je počet stavů v minimálním automatu. Nejvyšší časová složitost přidání stavu do registru je $O(log\ n)$. Při použití hashovací tabulky jako reprezentaci registru, je časová složitost těchto operací konstantní. Vyjádření maximální časové složitosti celého algoritmu je tedy $O(m*|w|*log\ n)$, zatímco průměrná časová složitost je $O(m*|w|)$.

3.2.2 Reprezentace konečného automatu

Velikost automatu a rychlost jeho použití závisí na metodě, která je použita pro jeho reprezentaci. Rozdíly mohou být velmi výrazné. Nemůžeme říct, že by nějaká metoda byla vhodná pro všechny typy použití a vykazovala nejlepší možné výsledky ve všech aspektech. Reprezentace automatu musí být volena jako kompromis mezi různými požadavky.

Jeden ze standardních postupů je vyjádření pomocí matice. Jedna osa je v matici vyjádřena čísly počátečních stavů a druhá osa čísly stavů koncových jednotlivých přechodů. Atributy stavů jsou uloženy jinde, například v dalším vektoru a položky v matici jsou vyjádřeny jako atributy přechodů. Tato metoda je neefektivní, protože při rozsáhlých automatech se vyskytují milióny stavů s velkým množstvím různých přechodů.

Další obvyklou možností je použití seznamů. Výhoda je, že pokud jsou přechody v paměti uloženy za sebou, není potřeba další paměť pro informaci o dalším přechodu. Nutný je ovšem údaj o počtu přechodů u jednotlivých stavů. Přechody mohou být uloženy jako za sebou jdoucí položky vektoru. Jestliže změníme vlastnosti, například počet přechodů, již nebude potřeba uchovávat stavy explicitně, budou uloženy implicitně v přechodech.

V použitém algoritmu je navíc využita komprese pro menší paměťovou náročnost. Spočívá ve spojování určitých stavů do jednoho za předpokladu, že oba tyto stavy mají shodně ohodnocené přechody. Pokud má tato metoda pracovat efektivně, musí být přechody setříděny. Typ třídění potom následně ovlivní i kompresi. Nejlepší výsledky se dosáhnou, pokud jsou přechody tříděny podle četností jejich výskytů.

Pro reprezentaci automatů existuje celá řada dalších metod, které jsou ve většině případů založeny na využití seznamů nebo tabulek s řídkým zaplněním.

4 Metody doplnění diakritiky

Různé algoritmy pro obnovu diakritiky vyžadují odlišná data. Většina z nich provádí jednoduché vyhledávání ve slovnících, a jestliže hledané slovo není nalezeno, použijí slova podobná. Stačí jim pouze seznam slov bez jakýchkoliv dalších upřesnění, ve kterém je slovo automatem přijato, pokud se nalézá v daném slovníku.

Pokročilejší algoritmy se již zabývají souvislostmi. Dochází k syntaktické analýze, která může být částečná nebo úplná. K ní jsou již zapotřebí morfologická data, která nemusejí být pro všechny jazyky lehce dostupná. Specifická forma dat pro morfologický slovník závisí na druhu automatů, který má být sestaven.

4.1 Učení na úrovni znaků

Tato metoda je představena v článku, jehož autorem je Rada F. Mihalcea [MIH-02] a ve kterém se objevuje nový pohled na analýzu obnovení diakritiky. Pokud bychom chtěli využít metody, zaměřující se spíše na analýzu slova, můžeme být v různých případech výrazně omezeni, protože:

- elektronické slovníky nejsou dostupné nebo je dostupná jen omezená velikost slovníků. Navíc, když slovník sám postrádá diakritiku, tyto metody přestávají být využitelné;
- nejsou dostupné nástroje pro morfologickou a syntaktickou analýzu;
- velikost archivních materiálů obsahující diakritiku je omezená. Tato velikost archivů dostupných na internetu následně ovlivňuje velikost slovníku, který může být vytvořen z těchto textů. Navíc autoři článků se v mnoha případech vyhnou používání diakritiky, z důvodů jednoduchosti, jednotnosti a v dřívějších dobách i nemožnosti správného kódování diakritiky;

Ve výše uváděném textu je zkoumána metoda pro obnovení diakritiky, která se zaměřuje na analýzu textu na úrovni znaků, místo na úrovni slova. Velká výhoda této metody je, že poskytuje prostředky pro zevšeobecnění mimo konkrétní slova. Algoritmus je schopen pracovat na vysoké úrovni i při malém množství tréninkových dat. V praxi to znamená, že upouštíme od pravidel zaměřených na slovní úroveň (například "*mas* má být doplněno na *máš*, když jej předchází zájmeno") a zajímáme se o zevšeobecnění na úrovni znaků (například "*a*, kterému předchází *m* a následují *bílé znaky*, se má změnit na *á*"). Tyto typy pravidel mají lepší použitelnost při užití malé slovní zásoby, kdy je v textu velké množství neznámých slov, nebo když nejsou k dispozici nástroje pro morfologickou a syntaktickou analýzu.

Je jasné, že písmena představují nejmenší možnou jednotku, na kterou je možné aplikovat jazykovou analýzu, a proto mají největší potenciál pro zevšeobecnění. Namísto toho, abychom uchovávali několik stovek tisíc slov, se kterými v algoritmech pracujeme (přibližná velikost slovníku

jazyka je kolem 150 000 výrazů), stačí mít jako vstup 26 znaků, představujících písmena abecedy. Metoda je pak zvláště určená pro jazyky, které postrádají veřejně dostupné zpracování textu nebo rozsáhlé elektronické slovníky s diakritikou. Navíc nevyžaduje jiné nástroje nebo zdroje, kromě korpusu slov s diakritikou a díky jednoduchosti algoritmu je rychlost zpracování velmi vysoká.

Pro testování byly zvoleny takové jazyky, které nejsou hojně používány ve světovém měřítku a následkem toho nemají mnoho veřejně dostupných nástrojů a zdrojů. Data byla získána z internetu hlavně z novinových archivů nebo z dostupných elektronických textů. Vznikl tak testovací korpus, který pro český jazyk obsahoval kolem 1.5 milionu slov.

Jelikož metoda pracuje na úrovni písmen, učící cílový atribut je představován nejednoznačnými písmeny, které se mohou v textu vyskytovat. Při testování nejsou uvažována velká písmena, která byla převedena na písmena malá. Pravidla, která se užívají v učebním algoritmu, mají výrazný vliv na konečnou přesnost. Nepoužijí se zde žádné morfologické nebo syntaktické analyzátory, ani se nespoléhá na předchozí a následná slova, protože by to mohlo mít za následek výskyt velkého množství neznámých slov. Jsou použity velmi jednoduchá pravidla, která se soustředí na předchozí a následná písmena, mezery, čárky, tečky a dvojtečky. Tyto znaky mohou ovlivnit proces učení a jeho přesnost je překvapivě velmi vysoká.

4.1.1 Vyhodnocení metody

Pro každý nejednoznačný pár písmen, procházíme text a generujeme všechny možné příklady, které se v korpusu vyskytují. Atributy jsou tvořeny *n* písmeny vlevo a vpravo od ne jednoznačného znaku a cílový atribut je nejednoznačný znak sám:

$$\mathbf{L}_{-n}\mathbf{;}\ \mathbf{L}_{-n+1}\mathbf{;}\ \mathbf{\ldots}\ \mathbf{;}\ \mathbf{L}_{-1}\mathbf{;}\ \mathbf{L}_{1}\mathbf{;}\ \mathbf{\ldots}\ \mathbf{;}\ \mathbf{L}_{n-1}\mathbf{;}\ \mathbf{L}_{n}\mathbf{;}\ \mathbf{L}_{0}$$

Pokud se podíváme na výsledky při doplnění diakritiky v tabulce 4.1, pro češtinu dosahuje tato metoda průměrnou přesnost 97.83%. Rozlišujeme správnost doplnění pro každé nejednoznačné písmeno a dále je uveden počet příkladů zjištěných z korpusu. Následně je vyčíslena přesnost doplnění, pokud byl vybrán pro přepis standardně nejčastěji se vyskytující znak z každého souboru nejednoznačných písmen. Tato hodnota je v kontrastu s přesností doplnění, která byla získána metodou, zaměřenou na učení aplikované pro jednotlivá písmena. Průměrná přesnost je dána velikostí vstupního korpusu dat. Při použití rozsáhlejší množiny dat se očekává zvýšení přesnosti výsledků. Zajímavé je, že míra přesnosti doplnění není přímo úměrná počtu nejednoznačných písmen v konkrétním jazyce. Například při testování byla úspěšnost v maďarštině horší, než v češtině, přičemž má maďarština pouze pět nejednoznačných sad, zatímco čeština jich má třináct.

Nejednoznačné skupiny písmen	Počet výskytů	Úspěšnost při nahrazení nejvíce pravděpodobným	Úspěšnost při analýze znaků
a á	649,886	75.01%	96.96%
c č	217,57	72.21%	97.08%
d ď	271,07	99.05%	99.86%
e é ě	768,051	74.59%	97.02%
i í	504,298	60.43%	96.29%
n ň	439,552	98.97%	99.71%
o ó	566,521	99.08%	99.86%
r ř	319,352	65.55%	97.60%
s š	380,805	84.44%	98.88%
t ť	387,214	99.05%	99.85%
u ú ů	264,408	80.89%	93.51%
y ý	191,317	65.55%	95.06%
z ž	219,082	66.49%	98.70%
Průměr		**80.44%**	**97.83%**

Tabulka 3.1 – Úspěšnost doplnění učením na úrovni znaků

Při pozorování se ukázalo, že křivka osvojování znalostí je na počátku velmi strmá a zvyšuje se s dalšími dostupnými daty. Na jisté hranici se učení zpomalí a pro každé procento zlepšení je potřeba velké množství vstupních dat. Bylo zjištěno, že nejlepší přesnost správného doplnění se vyskytovala při trénování pro deset obklopujících písmen (na každé straně pět).

Tato metoda je vhodná u jazyků, kde nejsou dostupné nástroje, pro analýzu tvarů slov. Také může být užitečná, pokud nejsou dostatečně obsáhlé korpusy dat a dochází k velkému výskytu neznámých slov. Pro naši práci využita nebyla, jelikož jsme měli k dispozici rozsáhlý korpus, který také obsahoval analýzu četnosti výskytu jednotlivých slov. Také tato metoda vykazuje nižší úspěšnost při doplnění diakritiky. Výsledné hodnocení se vztahuje na jednotlivá písmena, pokud bychom brali v úvahu celá slova byla úspěšnost ještě nižší.

4.2 Syntaktická a morfologická analýza

Další možností, jak postupovat při doplňování diakritiky, je analyzovat souvislost mezi jednotlivými slovy. Tady už je zapotřebí syntaktická analýza, pro kterou je nutné získat speciálně upravená morfologická data. Ve slovníku se pak rozlišují slova, která jsou tvořena ze slov základních, například řetězec *dělala* je forma (jedna z mnoha) slova *dělat.* Tyto řetězce pak odpovídají slovním jednotkám,

ke kterým jsou přidány další morfologické informace, které popisují vlastnosti slova. U slova *dělala* můžeme určit, že jde o sloveso v minulém čase, jednotného čísla, ve třetí osobě, vid nedokonavý. Pro uchování takového morfologického slovníku slouží konečné automaty. Slovo je děleno na dvě části – samotný tvar slova a komentář, který obsahuje základní tvar slova a jeho morfologické kategorie.

Tyto informace jsou pak využívány pro doplňování diakritiky v textu. Bere se úvahu, v jakém se dané slovo nachází kontextu, tzn. které slova a jakých tvarech se nacházejí před a za slovem. Důležitý praktický problém při zpracování představuje velikost zdrojových dat, která jsou při analýzách využívána. Systémy zpracování přirozeného jazyka (dále jen ZPJ), které zajišťují úplnou syntaktickou analýzu neomezených textů (např. elektronické slovníky), musejí obsahovat informace pro stovky tisíc slov.

4.2.1 Perfektní hashování

Pro vhodné uložení těchto dat byly vyvinuté techniky perfektního hashování založeny na konečných automatech, které umožňují velice kompaktní reprezentaci rozsáhlých slovníků s minimální dobou, která je potřebná k přístupu k datům. Pokud vycházíme z odborné práce Jana Daciuka [DAC3-02], kde je analyzován proces perfektního hashování, můžeme podrobněji zkoumat techniky, které jsou použity pro práci se slovníkem.

Uveďme, že hash funkce je transformace, která jako vstup přijímá řetězec znaků o libovolné délce a výsledkem je následně řetězec znaků s pevnou délkou, tzv. *hash* nebo také *otisk*. Hashování mapuje slova z daného souboru *n* slov na čísla (adresy). Jestliže je mapování bijekce do množiny *1…n*, mluvíme o perfektním hashování. Takto je možné vyčíslit všechna slova z jazyka přijatelného automatem. Vše, co je potřebné pro sestavení automatu využívající princip perfektního hashování je pouze jednoduchý seznam slov. Pro zachování pořadí slov v automatu by měl být tento seznam abecedně setříděný. Na klíče se díváme jako na slova z jazyka, který je akceptován automatem. Proto můžeme zkonstruovat automat, který tento jazyk akceptuje a uložit do něj všechna slova. Protože jazyk obsahuje konečné množství slov, automat musí být acyklický a pro zachování účinnosti také deterministický.

Dále potřebujeme stanovit mapování mezi slovy jejich čísly. Jestliže jsou všechny přechody v automatu setříděny, existuje takové mapování přímo v automatu. Slova jsou řazena podle jejich výskytu v automatu při využití metody postorder. Mapování může být tedy dáno počtem všech slov, která předcházejí dané slovo v automatu. To znamená, že dochází k procházení celého automatu, což je výpočetně velmi nepraktické. Nicméně je možné některé hodnoty předpočítat, takže hledání se může stát efektivním.

Příklad efektivního použití je uveden na holandském novinovém korpusu, kde bylo analyzováno asi 350 000 vět. Tento korpus obsahuje víc než 215 000 unigamů (samostatných slov), 1 785 000 bigramů (dvojice slov) a 3 810 000 trigramů. Pokud bychom chtěli reprezentovat všechny trigramy

v tomto korpusu čistě textově, budou zabírat více než 82MB paměti. Použitím obvyklé hash implementace (standardní knihovna C++), se zvýší během provádění operací použitá paměť na 362MB. Přitom inicializace hashe z tabulky trvá asi tři minuty. Pokud aplikujeme nové techniky perfektního hashování, je velikost použité paměti zredukována na pouhých 49MB a uložení kompaktního jazykového modelu zabere méně než půl sekundy.

V příkladu je demonstrováno, že velikost zdrojových dat, která jsou používány v ZPJ je velký problém. Také požadavky na operační paměť jsou vyšší. Při nalezení vhodného způsobu zpracování, budeme moci reprezentovat rozsáhlé jazykové modely velmi kompaktním způsobem za použití konečných automatů a dokážeme je vhodně využít například pro kontrolu pravopisu nebo doplňování diakritiky. Používání těchto modelů je mnohem rychlejší a v praxi nedochází k žádnému zpoždění při jejich aplikování.

4.3 Výběr na základě četnosti výskytu

Pro účel této aplikace je použita metoda vyhledávání všech ekvivalentů s diakritikou k výrazu bez diakritiky na základě definovaného slovníku. Volně dostupná implementace takovýchto technik je poskytnuta Janem Daciukem [DAC1-98]. Není nutné znát vztahy mezi jednotlivými slovy, stačí využít slovník, který obsahuje pouze jednotlivá samostatná slova. Tyto slova jsou uložena v konečném automatu, ve kterém se následně provádí jejich vyhledávání.

Pro správné vyhledání všech dostupných slov ze slovníku je potřeba definovat přepis, který určuje ekvivalenci písmen bez a s diakritikou (například *e* je možné přepsat na *é* a *ě*, ale také samozřejmě ponechat jako *e*. Pokud bychom měli výraz *jez*, začnou se v automatu vyhledávat všechny slova, které se mohou z tohoto výrazu vytvořit, na základě přepisu jednotlivých znaků. V našem případě se tety vyhledávají výrazy *jez, jéz, jéž, jěz, jěž, jež.* Pokud máme k dispozici korektní slovník českých slov, měla by být nalezena slova *jez* a *jež.*

Jestliže je několik variant doplnění diakritiky, musíme se rozhodnout, kterým slovem bude výraz nahrazen. V aplikaci jsme pro zpřesnění výsledného doplnění použili analýzu četnosti výskytu slov v textech. Spočívá v nahrazení výrazu slovem, které je v textu považováno za nejpravděpodobněji se vyskytující. Tato pravděpodobnost je uložena společně se slovem v automatu a následně porovnávána s hodnotami pravděpodobností všech slov, která byla nalezena jako vhodná pro nahrazení. Ostatní varianty jsou nabídnuty pro pozdější korekci uživatelem. Podrobněji je tato metoda popsána v následující kapitole společně s prezentací dosažených výsledků doplnění diakritiky na testovacím vzorku dat.

5 Vyhodnocení doplňování diakritiky v češtině

Dříve, než budeme moci vytvořit fungující a efektivní aplikaci, která bude doplňovat diakritiku v textu, měli bychom zjistit, s jakou úspěšnosti přepisu bude naše aplikace pracovat. V této kapitole si ukážeme různé postupy při korekci diakritiky, vyhodnotíme jejich úspěšnost a budeme se snažit určit takové vyhodnocování, které vykazuje nejvyšší pravděpodobnost správného přepsání. Pokud získáme uspokojivé výsledky, můžeme opustit oblast testování a pokusit se vystavět přístupnou aplikaci pro potřeby obyčejných uživatelů.

5.1 Analýza překladu a testovacích dat

Abychom mohli výsledky prohlásit za relevantní, měli bychom zvolit dostatečně objemnou množinu vstupních údajů, které budeme následně vyhodnocovat. K tomuto účelu nám poslouží testovací korpus dat, který byl získán z Laboratoře zpracování přirozeného jazyka. Připomeňme, že korpus je kolekce textových údajů v elektronické podobě. Používá se jako významný zdroj lingvistických dat a slouží ke zkoumání mnoha frekvenčních jevů jazyka. Pro tento účel jej také využijeme i my. Obsahuje soubor článků z běžného denního tisku a díky co nejobecnějšímu pokrytí uváděných textů se snažíme dosáhnout objektivních výsledků. Pokud bychom například uváděli pouze texty z určitých odborných oblastí, informace by byly nedostačující, zkreslené a nevypovídaly by o skutečném rozložení slov v textech.

V našem případě má testovací korpus deset milionů slov, což je považováno za dostačující počet pro uspokojivé výsledky. Tato slova jsou samostatně rozdělena po řádcích a pro následnou analýzu použita pouze ta, která jsou složena z písmen české abecedy. V takto vytvořeném souboru dat je potřebné zjistit četnost jednotlivých slov, abychom mohli určit pravděpodobnost, s jakou se vyskytuje konkrétní slovo s určitou diakritikou.

Analýza je prováděna v Bash Shellu v prostředí Linux a je založena na principu křížové validace. Ta spočívá v rozdělení testovacího vzorku dat na deset rovnoměrných částí. Z prvních devíti desetin korpusu se vytváří slovník, na který je následně aplikována zbylá desetina dat. Při tvorbě slovníku je z počátku určena četnost každého slova a vytvoří se unikátní seznam, kde je každé slovo zaznamenáno nejvýše jednou. Následně se provede abecední setřídění. Vytvoření slovníku vychází ze zjednodušeného unigramového jazykového modelu. To znamená, že pro celou skupinu slov, která se bez diakritiky píší stejně, je do slovníku vloženo pouze jedno slovo a to takové, které se v této části korpusu vyskytuje v největším počtu. Výraz bez diakritiky se na takové slovo přepíše vždy, bez

ohledu na význam nebo kontext věty. Při rovnosti množství výskytu je uloženo slovo, které je nejdříve v abecedě.

Příklad: Je zjištěno, že při vytváření slovníku z části korpusu se slovo ***celé*** vyskytuje zhruba v 72 procentech případů, slovo ***čele*** je obsaženo ve 26 procentech a slovo ***cele*** ve dvou procentech. Ve všech třech případech se jedná po odstranění diakritiky o stejný výraz ***cele***. Do vytvářeného unigramového modelu je ale uloženo pouze slovo ***celé***, protože má nejčetnější výskyt. Při následném doplňování bude tedy vždy výraz ***cele*** přepsán na ***celé***.

Takto vytvořený slovník z devíti desetin korpusu je uložen pomocí konečného automatu a stává se základem pro přidávání diakritiky pro zbylou desetinu testovacích dat. Slova v ní jsou zbavena diakritiky a porovnávána s výrazy ve slovníku. Pokud dojde k nalezení možnosti přepisu, je na výstupu uloženo zmiňované slovo ze slovníku. Hledaný výraz může být ve slovníku nalezen nejvýše jednou, jelikož se jedná o unigramový model. Jestliže je shodný s původním slovem před zbavením diakritiky, je překlad slova hodnocen jako úspěšný, v opačném případě jako neúspěšný. Takto se opakovaně vyhledá ve slovníku možná varianta překladu ke každému slovu z testovacích dat. Pokud dané slovo není možné žádným způsobem přeložit, je ke slovu uložena informace, že nebyl nalezen odpovídající překlad.

Příklad: Navážeme na příklad z minula. Už jsme si vysvětlili, proč je v našem unigramovém modelu uloženo pouze slovo ***celé*** a nikoliv i další, bez diakritiky shodné výrazy**.** Nyní si představme, že ve vstupním vzorku dat, která se porovnávají s vytvořeným slovníkem, se vyskytne slovo ***celé*** 5krát, slovo ***čele*** 3krát a slovo ***cele*** 2krát. Tato slova zbavíme diakritiky a zkusíme zjistit její opětovné doplnění pomocí slovníku. Ve všech případech dojde k doplnění na ***celé,*** protože pouze tento výraz se nachází ve slovníku. Následným vyhodnocením překladu zjistíme, že jsme měli v tomto případě, pokud bychom uvažovali jen zmíněná slova, je úspěšnost pouze 50%. Ve skutečnosti je ale toto číslo daleko vyšší.

Po provedení první etapy testu se vyhodnotí úspěšnost překladu a následně dojde k rotaci částí, ze kterých je vytvářen slovník, a části, ze které se berou vstupní data (tzn. pokaždé je vyhodnocována jiná část dat a tato část není obsažena ve vytvořeném slovníku). Celý postup je opakován desetkrát a ze všech vyhodnocení vypočten průměr úspěšnosti automatického doplnění diakritiky do textu na daném vzorku dat.

5.2 Postupné zpřesňování překladu

Naším cílem v této fázi je nastolit takový mechanismus překladu, aby jeho úspěšnost byla statisticky co nejvyšší. V prvotní chvíli dochází k doplňování diakritiky automaticky bez zásahu uživatele, jeho korekce a zpřesnění výsledků přijde na řadu později. V Tabulce 5.1 je tedy uvedena základní úspěšnost doplnění diakritiky pro náš testovací korpus, kdy jsou slova hledána ve slovníku vytvořeném ze zbylé části korpusu.

	Slova na překlad	Správný překlad	%	Špatný překlad slov ze slovníku	%	Není ve slovníku	%
1. část	705249	657437	93,22	27883	3,95	19929	2,82
2. část	677279	635665	93,85	22785	3,36	18829	2,78
3. část	686586	644662	93,89	23078	3,36	18846	2,74
4. část	670821	628329	93,66	26333	3,92	16159	2,40
5. část	613595	576171	93,90	21415	3,49	16009	2,60
6. část	653287	610434	93,44	23977	3,67	18876	2,88
7. část	669814	626816	93,58	25660	3,83	17338	2,58
8. část	667081	623137	93,41	25365	3,80	18579	2,78
9. část	676148	632896	93,60	26287	3,88	16965	2,50
10. část	657967	611222	92,89	27570	4,19	19175	2,91
Celkem	**6677827**	**6246769**	**93,54**	**250353**	**3,74**	**180705**	**2,70**

Tabulka 5.1- Úspěšnost překladu - verze 1.1

Komentář k tabulce:

- **Slova na překlad –** takové výrazy, které jsou charakteristické pro české slovo, obsahují písmena české abecedy. Vyřazena jsou slova obsahující znaky jiných abeced, čísla matematické výrazy apod.
- **Správný překlad –** slova, která se pomocí vytvořeného slovníku přeložila shodně, jako originál
- **Nesprávný překlad –** chybně doplněné výrazy, které se neshodují s původním tvarem. Ve slovníku se nachází četnější slovo, které je bez diakritiky shodné s právě překládajícím.
- **Není ve slovníku –** překládaný výraz nebyl nalezen ve slovníku, neexistuje k němu známý předpis pro doplnění diakritiky

Průměrná hodnota úspěšného překladu byla v tomto případě vyčíslena na 93,54%. S tímto údajem se ale nespokojíme a budeme se jej nadále snažit zpřesňovat. Nezanedbatelnou část dat tvoří

výrazy, které nejsou nalezeny ve slovníku, a tudíž nevíme, jak diakritiku vůbec doplnit. Do správného překladu jsme je nezapočítali, přitom je ale velké množství těchto slov bez diakritiky, a tudíž ve správném tvaru. Můžeme tedy v další verzi 1.2 tato slova bez diakritiky, která nebyla nalezena ve slovníku, považovat jako úspěšně přeložená. Nadále ovšem musíme počítat s tím, že pro neznámá slova, která diakritiku obsahují, nemáme žádný vzor a nemůžeme je správně přeložit.

	Slova na překlad	Správný překlad	%	Špatný překlad	%	Není ve slovníku	%
1. část	705249	666697	94,53	27883	3,95	10666	1,51
2. část	677279	643823	95,04	22785	3,36	10771	1,59
3. část	686586	652517	95,03	23078	3,36	10991	1,60
4. část	670821	637416	95,02	26333	3,92	7067	1,05
5. část	613595	582558	94,94	21413	3,48	9621	1,56
6. část	653287	618982	94,75	23977	3,67	10320	1,57
7. část	669814	635178	94,82	25659	3,83	8976	1,34
8. část	667081	631945	94,73	25365	3,80	9771	1,46
9. část	676148	640768	94,76	26287	3,88	9093	1,34
10. část	657967	620972	94,37	27567	4,18	9428	1,43
Celkem	**6677827**	**6330756**	**94,80**	**250353**	**3,74**	**96704**	**1,44**

Tabulka 5.2 - Úspěšnost překladu - verze 1.2

V této fázi se nám podařilo zpřesnit překlad na 94,80%. Stále ovšem pro sestavení slovníku používáme pouze výrazy, které jsou k dispozici v testovacím korpusu. Samozřejmě, že v něm nejsou zdaleka obsažena všechna česká slova a slovní tvary, které se ať už v mluveném nebo v psaném projevu vyskytují. Proto jsme pro další zpřesnění přidali k vytvořenému slovníku z části korpusu i souhrnný slovník všech českých slov, který byl k dispozici opět z Laboratoře pro zpracování přirozeného jazyka. Obsahuje více než 6 milionů slov, která postihují většinu spisovné slovní zásoby českého jazyka. Pokud jsme tedy hledali i v tomto slovníku, povedlo se nám velmi výrazně zvýšit přesnost doplnění až na 96,23%, jak ukazuje tabulka 5.3. Pro překlad jsme používali jen takové výrazy, které se nalézají aspoň v jednom slovníku, takže se nestane, že by dané slovo nebylo nalezeno. Nenalezené výrazy necháváme nerozhodnutelné. Připomeňme jen, že nesprávný překlad vzniká tím, že se překládané slovo vyskytuje v korpusu méně často než slovo, které bylo použito při vytvoření slovníku.

	Slova na překlad	Správný překlad	%	Špatný překlad	%
1. část	664313	637643	95,98	26670	4,01
2. část	646833	624170	96,49	22663	3,50
3. část	656387	633323	96,48	23064	3,51
4. část	616580	592745	96,13	23835	3,86
5. část	584209	563694	96,48	20515	3,51
6. část	618070	594836	96,24	23234	3,75
7. část	627499	602866	96,07	24633	3,92
8. část	620098	595990	96,11	24108	3,88
9. část	634255	609311	96,06	24944	3,93
10. část	596029	574134	96,32	21895	3,67
Celkem	**6264273**	**6028712**	**96,23**	**235561**	**3,76**

Tabulka 5.3 - Úspěšnost překladu - verze 1.3

5.3 Odchylka četnosti slov

Až do této fáze byly všechny překlady prováděny automaticky na základě unigramového modelu použitého při vytváření slovníku. Vše vycházelo ze zjištění četnosti slov a vybrání nejpravděpodobnějšího tvaru, na který je hledaný výraz možno přepsat. Uživatel v tomto případě neměl do rozhodování možnost zasahovat. Správnost překladu se jeví sice poměrně vysoká, ale přesto se vyskytuje relativně velké množství výrazů, u kterých je diakritika doplněna chybně. Pro další zvýšení úspěšnosti překladu je tedy vhodné zvolit určitou míru poloautomatizace.

Znamená to, že pokud je při výběru správné varianty určitá blíže specifikovaná procentuální možnost chyby, ponechá se sporné rozhodnutí na uživateli. Ten podle kontextu a smyslu textu dokáže tyto výrazy korektně přepsat podle nabízených možností. V praxi se jedná nejčastěji o takové výrazy, kdy je četnost výskytu stejných slov píšících se bez diakritiky shodná, nebo se jejich odchylka liší jen velmi mírně. Tuto odchylku je možné postupně měnit a analyzovat míru úspěšnosti překladu a zásahu uživatele. V tomto případě platí úměra, že pokud chceme nadále zpřesňovat překlad textu, musíme zvyšovat počet sporných slov, která nejsou rozhodována automaticky. To znamená, že pro správnou efektivnost a účelnost celé aplikace je nutné zvolit optimální poměr mezi automatizací překladu a zásahem uživatele. Při vytváření slovníku se již opouští od unigramového modelu a jsou již do něj ukládány všechna slova i s odchylkou četnosti přepisu. Tím umožníme tyto informace dále využívat a dokážeme nastavit určitou přesnost doplnění.

Příklad: Při získávání dat z korpusu je zjištěno, že slovo **sraz** se vyskytuje 856krát a slovo **sráz** 58krát**.** Do slovníku jsou uloženy obě slova i s procentuální odchylkou četnosti, která mezi nimi činí 93,2%. Z uvedených údajů vyplývá, že v praxi je tato odchylka přepisu natolik vysoká, že rozhodování uživatele není potřeba a je ponecháno pouze automatické doplnění.

Určitá poloautomatizace vyhodnocování ovšem není vhodná pouze pro zpřesnění doplňování diakritiky. Při vyhodnocování se vychází z korpusu, který při své velikosti nemusí objektivně zachycovat četnost jednotlivých slov. U méně frekventovaných výrazů s jedním nebo dvěma výskyty v korpusu nemůže být statisticky přesně stanovena odchylka četnosti. Proto ponecháváme na uživateli rozhodování i o slovech, která přicházejí v úvahu pro přepis, u kterých je četnost menší nebo rovna třem. V opačném případě by docházelo ke zkreslení výsledků, které se mohou od skutečnosti odlišovat.

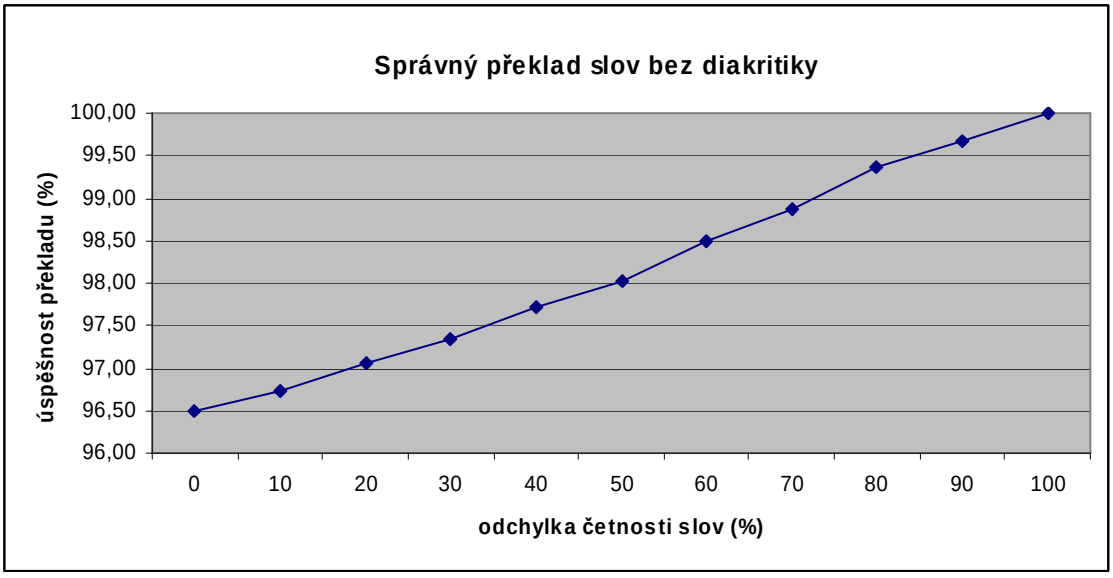
Příklad: Uvažujeme, že se v korpusu slovo **rozbedni** vyskytuje dvakrát a slovo **rozbední** pouze jednou. Odchylka četnosti by v tomto případě byla 50%. Počet výskytů je ale natolik malý, že se v této odchylce nemusí odrážet reálný poměr četností těchto slov. Proto výsledné doplnění není prováděno automaticky.

Při následující analýze postupně vzrůstá hodnota odchylky četnosti slov, která mají shodný zápis bez diakritiky. Pokud je odchylka u daného slova nižší nebo rovna než stanovená hodnota, rozhoduje o správnosti přepisu uživatel. V tabulce 5.4 jsou uvedeny průměrné údaje pro celou křížovou validaci. Pro odchylku 0% uživatel rozhoduje pouze slova, která mají v korpusu stejný počet výskytů, a slova, jejichž četnost je pro každou z více možností menší nebo rovna 3. Automatizace je v tomto případě nejvyšší, ale za cenu méně korektního doplňování diakritiky u sporných výrazů. Naopak při odchylce 100% rozhoduje uživatel pokaždé, když se nabízí více variant překladu. Nedochází zde k nesprávnému automatickému překladu, protože jsou doplněna ta slova, u kterých je pouze jedna varianta přepisu. Při postupném zvyšování odchylky je patrné, že úspěšnost překladu roste, ale za cenu menší automatizace a větší námahy uživatele. V další analýze se budeme snažit nalézt vhodný poměr mezi těmito argumenty.

odchylka (%)	Slova na překlad	Správný překlad	%	Rozhodne uživatel	%	Špatný překlad	%
0	6264273	6044564	96,49	11423	0,19	219709	3,51
10	6264273	6059472	96,73	42415	0,70	204801	3,27
20	6264273	6079460	97,05	85615	1,41	184813	2,95
30	6264273	6098182	97,34	130951	2,15	166091	2,65
40	6264273	6120697	97,71	184630	3,02	143576	2,29
50	6264273	6141095	98,03	239759	3,90	123178	1,97
60	6264273	6169473	98,49	329208	5,34	94800	1,51
70	6264273	6193198	98,87	418105	6,75	44413	1,13
80	6264273	6224598	99,37	567330	9,11	39675	0,63
90	6264273	6243482	99,67	701321	11,23	20791	0,33
100	6264273	6264273	100,00	1836206	29,34	0	0,00

Tabulka 5.4 - Úspěšnost překladu - verze 2.0

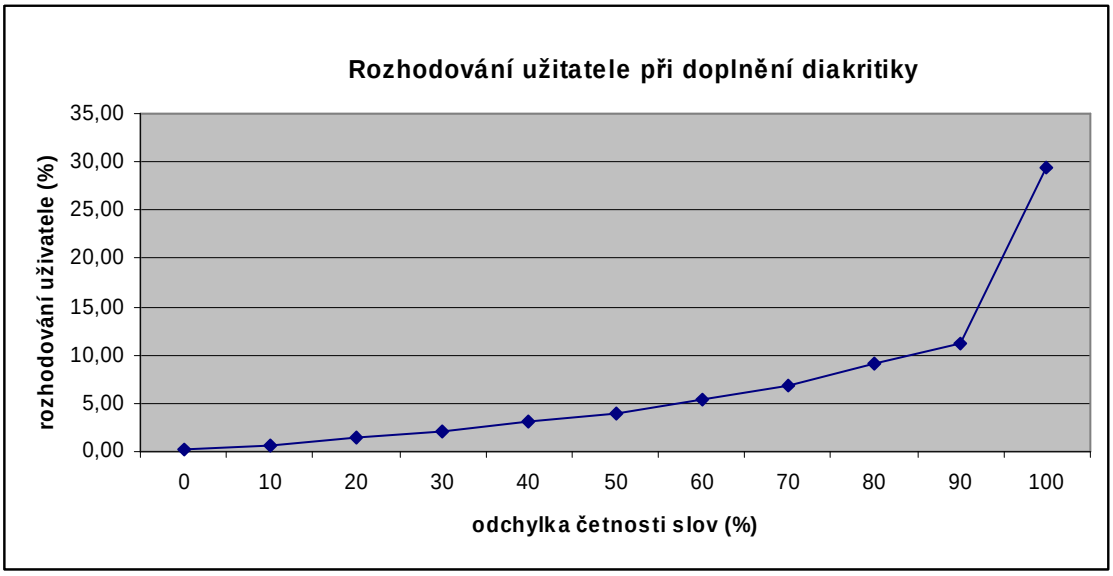
Blíže se podíváme, jaký vztah existuje mezi zvyšováním odchylky (tzn. posunování hranice pro automatické rozhodování) a úspěšností překladu. Závislost je vynesena v Grafu 5.1. Při pohledu na graf se dá usuzovat, že úspěšnost správného doplnění diakritiky roste lineárně a nejsou patrné žádné větší výkyvy. Při užívání celé aplikace se budeme snažit nalézt optimální hodnoty mezi těmito veličinami.



Graf 5.1 – Správný překlad slov bez diakritiky

Druhý graf znázorňuje procento slov, o jejichž správnosti se nerozhoduje automaticky, ale z více možností si vybírá uživatel. Opět je sledováno narůstající procento v porovnání se vrůstající odchylkou četnosti slov. Můžeme říct, že zde má graf mírně exponenciální průběh. Vyplývá z toho, že

při velkých odchylkách je množství slov na rozhodování v poměru daleko větší, než při odchylkách malých. Velký skok při sto procentech je dán tím, že uživatel se zabývá všemi výrazy, kde je možné uvést několik variant. Protože testovací data vychází z běžného textu s překlepy a gramatickými chybami, objevují se zde i výskyty takových slov, které nejsou gramaticky správně. I když je jejich četnost zanedbatelná, uživatel se jimi při takto stanovené odchylce přesto musí zabývat. V praxi je zřejmé uplatňovat odchylku mezi 70% – 90% kdy možnost zásahu uživatele se pohybuje kolem 10% a chybný překlad není vyšší než 0,5%.



Graf 5.2 – Rozhodování uživatele při doplnění diakritiky

Na závěr této kapitoly si v tabulce ukážeme, do jaké míry se nám podařilo zlepšit správnost doplňování diakritiky do textu. Z počátku jsme uvažovali automatický aparát nezávislý na jiných vlivech. Postupně jsme zapojili poloautomatické rozhodování, které nám dokázalo navýšit úspěšnost překladu až na ideálních 100%, ale za cenu ruční korekce u slov s několika možnostmi přepisu. Zachyceny jsou také relativní zpřesnění přepisu proti první a předcházející verzi.

	Správný překlad	Nesprávný překlad	Relativní zlepšení proti první verzi (%)	Relativní zlepšení proti předcházející verzi (%)
Verze 1.1	93,54	6,44		
Verze 1.2	94,80	5,18	19,57	19,57
Verze 1.3	96,23	3,76	41,61	27,41
Verze 2.0 - 0%	96,49	3,51	45,50	6,65
Verze 2.0 -10%	96,73	3,27	49,22	6,84
Verze 2.0 - 20%	97,05	2,95	54,19	9,79
Verze 2.0 - 30%	97,34	2,65	58,85	10,17
Verze 2.0 - 40%	97,71	2,29	64,44	13,58
Verze 2.0 - 50%	98,03	1,97	69,41	13,97
Verze 2.0 - 60%	98,49	1,51	76,55	23,35
Verze 2.0 - 70%	98,87	1,13	82,45	25,17
Verze 2.0 - 80%	99,37	0,63	90,22	44,25
Verze 2.0 - 90%	99,67	0,33	94,88	47,62
Verze 2.0 100%	100,00	0,00	100,00	100,00

Tabulka 5.5 – Relativní zlepšování úspěšnosti překladu

Samotné stanovení optimální odchylky bude dále zkoumáno na vytvořené aplikaci. U teoretického testování a měření úspěšnosti jsme dosáhli uspokojivých výsledků. Tyto poznatky se budeme dále snažit potvrdit a upřesnit na koncových uživatelích. Bude zjišťována prospěšnost, rychlost a míra efektivity, s jakou se nám doplňování diakritiky podaří realizovat.

6 Návrh a implementace aplikace

Následující část je věnována shrnutí praktické části této práce. Nejprve se podíváme na požadavky, které byly vytyčeny jak zadáním, tak získané v průběhu práce. V pokročilejších fázích vývoje se tyto požadavky nadále zpřesňují. Dále je zachycen postup při vytváření slovní zásoby, která je následně využívána. Budeme se také zabývat návrhem a implementací samotné aplikace pro doplňování diakritiky, shrneme dosažené výsledky a na závěr budeme diskutovat přínos, který tato aplikace má.

6.1 Specifikace požadavků

Pokud chceme vytvořit fungující aplikaci umožňující efektivní práci, je potřeba před vlastní implementací provést analýzu požadavků, které jsou na její tvorbu kladeny. Abychom mohli diakritiku úspěšně doplňovat, je potřeba vycházet z určitého množství statistických údajů, ze zachycení četnosti slov v textech a samozřejmě i ze všech dostupných tvarů českých slov. K tomuto účelu je nutné vytvořit kompaktní slovník používaných výrazů, ve kterém budou procentuálně zachyceny také pravděpodobnosti doplňování diakritiky.

Po vytvoření slovníku je dalším úkolem sestavit ukázkovou aplikaci, která by tento slovník využívala. Tato aplikace má umožnit interaktivní doplňovaní diakritiky, jejím cílem je maximální usnadnění a urychlení činnosti. Uživatel by měl mít možnost snadno a rychle vybírat z nabídky slov, je-li k jednomu slovu bez diakritiky více možných slov s diakritikou. Díky možnostem, které umožňují samotné nastavení programu, si uživatel může přizpůsobit chování aplikace vlastním potřebám a například vhodně zvolit kompromis mezi automatickou a poloautomatickou činností.

Požadavky na aplikaci se dají shrnout do následujícího seznamu:

- vytvoření slovníku - z dostupných statistických údajů vytvořit vhodný slovník použitelný pro vyhodnocování přepisu diakritiky v aplikaci.
- doplnění diakritiky – je zde důležitá efektivita, rychlost a korektnost, s jakou je doplňování prováděno. Aplikace by měla dávat na výběr několik nebo všechny varianty přepisu a při velké pravděpodobnosti správného přepsání se rozhodovat za uživatele. Ovládání by mělo být intuitivní.
- odstranění diakritiky – jednoduché nahrazení písmen s háčky a čárkami jejich ekvivalenty bez diakritiky. Vše je prováděno podle předem stanoveného neměnného předpisu vycházejícího ze souboru znaků.
- uživatelské rozhraní – jednoduché ovládání aplikace myší i klávesnicí. Tento způsob umožňuje maximálně efektivní práci. Dále přehledný vzhled grafických prvků spolu s nápovědou a dalšími užitečnými informacemi.

- zachování opravených textů - možnost načítat původní text pro korekci ze souboru a také ukládat následně opravený text do souboru
- přenositelnost – aplikace je navrhována tak, aby ji bylo možné využívat v operačních systémech Windows a Linux

Pro upřesnění a detailnější vysvětlení důležitých požadavků se blíže podíváme na některé konkrétnější specifikace při vytváření aplikace. Abychom vůbec mohli diakritiku doplňovat, je třeba sestavit vhodný slovník, ze kterého budeme dále vycházet.

6.1.1 Vytvoření slovníku

Pro vytvoření dostatečně přesného slovníku, ze kterého bychom mohli úspěšně čerpat informace pro doplňování diakritiky, potřebujeme velké množství vstupních dat. Z takových údajů budeme následně schopni přesně určovat a posuzovat míru pravděpodobnosti přepisu pro každé jednotlivé slovo. V našem případě jsme vycházeli z dat, která obsahují několik desítek milionů slov získaných z běžného textu v denním tisku a na internetu. Tyto údaje jsme dále upravovali, abychom v konečné podobě docílili souhrnných a použitelných informací, kterých jsme následně využívali při vyhodnocování. Nejprve jsou data zredukována, aby se každý výraz v tomto seznamu vyskytoval právě jednou a u každého z nich byla jako další informace uvedena četnost, s jakou se v souboru dat vyskytuje.

Z těchto výrazů jsme museli vybrat pouze taková, která jsou pro naší potřebu dostačující, tzn. vyhovující české abecedě, neobsahující číslice, prázdné znaky, atd. Všechna taková slova byla převedena na malá písmena a znovu sloučena do unikátního seznamu s četností. Tato četnost byla u všech slov zvýšena o jedna. Je to z toho důvodu, že k těmto datům byl také přidán soubor všech spisovných českých slov a zvýšením četnosti upřednostňujeme ta slova, která se vyskytla v běžném textu. Tudíž existuje vyšší míra pravděpodobnosti přepisu na právě taková slova.

K četnosti byla následně i přidána informace o tvaru slova po odstranění diakritiky a podle tohoto údaje byl celý slovník setříděn. Pro všechny výrazy, které se píší bez diakritiky, shodně byla vypočtena pravděpodobnost přepsání pro každé jedno slovo.

Příklad: data obsahují údaje:

cele celé 143433; **cele čele 54009;** **cele cele 4828;**

tyto tři výrazy tvoří samostatnou skupinu, ve které je každé slovo po odstranění diakritiky psáno shodně. Do výsledného slovníku jsou uloženy pouze údaje o pravděpodobnosti přepisu pro každé jedno slovo:

celé 71,8%; **čele 25,9%** **cele 2,3%**

Z důvodů ušetření paměťových nároků jsou údaje o slově a pravděpodobnosti vkládány do slovníku bezprostředně za sebou ve tvaru *"slovo""číslo"*, kde číslo je uloženo na třech znacích a reprezentuje procentuální možnost přepisu s přesností na desetiny bez desetinné tečky. V našem případě by bylo zapsáno: **celé718**. Tento způsob ukládání slovníku není nejefektivnější co se týká paměťových nároků, ale v našem případě dostačující.

Takto vytvořený slovník se uloží do struktury trie pomocí nástrojů pro práci s konečnými automaty Jana Daciuka. Tato metoda je vhodná pro následné vyhledávání a její využití je diskutováno v předcházejících kapitolách. Po uložení zabírá výsledný slovník pouze okolo 10MB paměti, což znamená výrazné snížení paměťových nároků, jelikož vstupní data zabírala paměť zhruba o velikosti 150MB.

6.1.2 Doplnění diakritiky

Na doplňování diakritiky jsou v naší práci kladeny nejvyšší nároky. Je to stěžejní činnost celé aplikace, a tudíž musíme klást důraz na to, aby byla provedena co možná nejlépe. Uživatel má na výběr několik možností nastavení chování programu, které mu umožňují zrychlit a zefektivnit svoji práci. Může tak docílit stoprocentní úspěšnosti automatického překladu nebo naopak velmi rychlého doplnění s připuštěním malého množství chyb. Při používání aplikace mohou uživateli pomoci také následující možnosti:

- **automatické rozhodování** – určuje se procentuální hranice úspěšnosti překladu, nad kterou je překlad proveden automaticky bez zásahu uživatele a považuje se za správný. V opačném případě jsou nabídnuty všechny možnosti, přicházejí v úvahu pro přepis. Pokud by automatická korekce nebyla správná, je možná její pozdější ruční oprava
- **skrytí málo pravděpodobných slov** – naše data vycházejí z reálných textů, ve kterých se objevují překlepy, přepisy slov nebo slova s neznámým významem. Jejich četnost je v porovnání s ostatními výrazy velmi malá, přesto se objevují v nabízených možnostech pro doplnění. Tomu se může zabránit určením procentuální hranice, za kterou se již výrazy budou považovat za chybné a nebudou se zobrazovat při výběru možností. Tato volba zjednodušuje a zpřehledňuje práci s aplikací
- **automatická propagace výběru –** pokud upravujeme text, ve kterém se vyskytuje velmi často slovo, které má menší pravděpodobnost přepisu, a tudíž nebude nabízeno na prvním místě, musíme ho stále dokola opravovat. V takovém případě si překlad zjednodušíme tím, že povolíme dále šířit prvotně vybraný výraz. Pro všechna další taková slova v textu bude nabízen přepis, který jsme zvolili při prvním výskytu
- **zvýraznění slov, která nejsou ve slovníku** – přes velké množství dat, použitých pro tvorbu slovníku, může nastat případ, že hledané slovo nebude ve slovníku nalezeno. Takové slovo

bude zvýrazněno a uživatele bude informovat, že mohlo dojít k překlepu, nebo že se v takovém výrazu může vyskytovat diakritika, kterou není možné automaticky doplnit.

6.2 Programové vybavení

Pro dosažení stanovených požadavků jsem jako implementační jazyk zvolil Javu. Tento jazyk je charakterizován jako jazyk třetí generace, imperativní jazyk vysoké úrovně. Není určen výhradně pro specifickou aplikační oblast, můžeme tedy mluvit o jeho univerzálnosti. Jedná se o jazyk objektově orientovaný, to znamená, že výpočet je realizován jako volání metod a zasílání zpráv objektů. Největším přínosem Javy je bezesporu plná přenositelnost programů na úrovni zdrojového a také přeloženého kódu na libovolnou platformu bez nutnosti jejich rekompilace. Programy se totiž nepřekládají do strojového kódu konkrétního procesoru, ale do nezávislé podoby, tzv. bytového kódu. Tento kód pak může být interpretován na jakémkoliv počítači. Kompatibilita je tedy zajištěna na binární úrovni. Jazyk obsahuje také třídy pro návrh formulářů (GUI), které byly pro můj projekt potřeba.

Jazyk Java obsahuje i třídy pro práci s externími programy, dokáže číst/zapisovat standardní vstup/výstup těchto programů, nic tedy nebránilo použít Javu jako implementační jazyk. Z dostupných prostředí pro tento jazyk jsem si zvolil prostředí NetBeans 5.5.

O samotné ukládání a prohledávání slovníku se v mé aplikaci starají externí nástroje pro efektivní ukládání slovníku vyvinuté Janem Daciukem. Pro jejich naprogramování byl použit objektový jazyk C++ a zdrojové texty jsou volně dostupné a přeložitelné pro různé platformy.

Pro původní testování úspěšnosti překladu před vytvořením samotné aplikace byly použity sady skriptů, vytvořené v příkazovém interpretu bash shell pod operačním systémem Linux.

Pro vytvoření komplexního slovníku byla použita data obsahující seznam používaných českých slov společně s jejich četností. Údaje byly poskytnuty z Laboratoře zpracování přirozeného jazyka.

6.3 Implementace

Tato kapitola se ve stručnosti věnuje vlastní implementaci aplikace. Pro implementaci byl zvolen objektově orientovaný přístup, proto kapitola popisuje implementační třídy a jejich klíčové atributy a funkce.

6.3.1 Třídy Word a Words

Pro potřeby aplikace bylo výhodné zpracovávat vstupní text po slovech, pro tyto účely existují třídy Word a Words. Jedno slovo v rámci textu je určeno objektem Word, ten obsahuje index počátku a konce slova v řetězci který udává celý text. Třída Words pak reprezentuje celý text a je vždy postavena na konkrétním řetězci. Některé důležité metody shrnuje tabulka:

Words(String text)	Konstruktor, inicializuje interní struktury, volá metodu processText()
processText()	Projde vstupní text, rozdělí ho na slova a naplní kolekci slov (instance třídy Word) a kolekci různých slov
getWords()	Vrací seznam všech slov, tak jak jdou po sobě v původním textu
getWordsUnique()	Vrací seznam různých slov, které se vyskytují v textu.

6.3.2 Třída DiacriticWord

Tato třída je jednoduchý obal základního typu String s tím, že mu přiřazuje i pravděpodobnost pro toto slovo. Jedná se o pomocnou třídu, která kromě jednoduchých přístupových metod obsahuje již jen metodu pro porovnání dvou objektů DiacriticWord a funkci převádějící objekt této třídy do řetězce. Objekty této třídy jsou používány při vracení pole slov (různé možnosti slova s diakritikou pro slovo bez diakritiky). Pole těchto objektů lze pak jednoduše řadit podle pravděpodobnosti.

6.3.3 Třída WordAlternative

Třída WordAlternative reprezentuje jedno slovo bez diakritiky a k němu odpovídající seznam možných slov s diakritikou. Dále obsahuje vazbu na slovo v rámci textu (pozice počátku a konce slova) a umožňuje slovo zvýraznit zvýrazňovačem libovolné barvy. Samozřejmostí jsou přístupové funkce pro nastavování a zjišťovaní parametrů. Významné metody shrnuje tabulka:

highlightActive()	Zvýrazní toto slovo jako aktivní, toho je dosaženo změnou barvy pozadí slova v textovém poli.
highlightNormal()	Základní zvýraznění slova, indikuje že existuje více variant slov s diakritikou a je na uživateli aby opravil případnou chybu.
highlightNotFound()	Zvýrazní slovo, které nebylo nalezeno ve slovníku
getAlternatives()	Tato metoda vrátí pole možných variant (instance třídy WordAlternative) s diakritikou.
setSelected(int)	Nastaví vybranou alternativu, volá se jakmile uživatel vybere ze seznamu slov některou z variant.

6.3.4 Třída WordAlternatives

Tato třída navazuje na předchozí. Představuje obal pro seznam objektů WordAlternative a reprezentuje tak text bez diakritiky rozdělený na jednotlivá slova, přičemž u každého slova máme k dispozici i informace o možných odpovídajících slovech s diakritikou. Mezi klíčové metody patří jumpToNextAlternative() a jumpToPrevAlternative() které slouží pro pohyb mezi slovy v textu. Tyto

metody ignorují slova, u kterých nejpravděpodobnější varianta přesahuje nastavenou mez a provádějí i odpovídající změny ve zvýraznění, aktivní slovo je vždy jen jedno a je zvýrazněno odlišně od ostatních slov. Mezi další důležité funkce patří metoda highlightAll() která provede zvýraznění všech slov podle nastavené meze pravděpodobnosti, tato mez se může změnit za běhu a metody pro pohyb po slovech i pro zvýraznění musí tuto změnu reflektovat.

jumpToNextAlternative() jumpToPrevAlternative()	Metody pro pohyb pomyslného kurzoru po zvýrazněných slovech (tedy potencionálně chybně doplněných). Zajišťují zvýraznění slova a inicializace seznamu obsahujícího různé možnosti doplnění diakritiky.
jumpByWordStart(int)	Pohyb po slovech s pomocí myši zajišťuje tato metoda. Podle pozice na kterou se přesunul textový kurzor (tedy kam bylo kliknuto) zjistí které slovo se pod kurzorem nachází a provede podobné akce jako předchozí metody.
highlightAll()	Projde celý text a v závislosti na nastavení zvýrazní všechna slova která to vyžadují. Tato metoda se volá například po změně nastavení, kdy se mohla změnit mez pravděpodobnosti pro zvýraznění slov.
addAlternative()	Vloží do interní struktury další slovo, využívá se při inicializaci, na počátku doplněni diakritiky.

6.3.5 Třída DiacriticBuffer

DiacriticBuffer je klíčovou třídou celé aplikace. Zajišťuje samotné dotazy na slovník a vyhledává tak ke slovům bez diakritiky odpovídající slova s diakritikou včetně jejích pravděpodobností. Třída tak vlastně slouží jako objektový obal pro externí program fsa_accent, který je v této aplikaci využíván. Konstruktor třídy zjišťuje, zda jsou dostupné všechny potřebné soubory, jedná se hlavně o samotný program fsa_accent, dále o vytvořený slovník a pomocné soubory. Bez těchto prostředků by nebylo možné diakritiku doplňovat, v takovém případě dojde k vyvolání výjimky.

Hlavní metodou této třídy je buildMap(), jejímž parametrem je celý text bez diakritiky (reprezentováno objektem Words). Tento text je rozdělen na jednotlivá slova a ty jsou následně vložena do seznamu tak, aby v něm každé slovo bylo právě jednou. Poté se spustí externí program fsa_accent a jeho standardnímu vstupu je předán seznam slov. Z výstupu programu se pak čte odpověď, která pro každé zapsané slovo obsahuje seznam slov s diakritikou i s pravděpodobnostmi. Výsledek se ukládá do struktury HashMap, ke každému slovu bez diakritiky je přiřazen seznam slov s diakritikou. Mezi další metody této třídy patří prohledávaní tabulky postavené funkcí buildMap a vracející seznam slov i pravděpodobnosti výskytu.

buildMap(Words)	Inicializace, volání externího programu fsa_accent a zpracování jeho výstupu, uložení do interní hašovací tabulky.
getDiacriticWords(String)	Přístup do interní tabulky, pro zadané slovo bez diakritiky vrací seznam všech možností doplnění diakritiky i s odpovídajícími pravděpodobnostmi.

6.3.6 Třída DiacriticRestorer

Třída reprezentující samotný doplňovač diakritiky. Využívá služeb většiny dříve popsaných tříd a obsahuje i vazby na grafické uživatelské rozhraní. Zajišťuje obsluhy událostí které vyvolá uživatel. Některé významné metody shrnuje tabulka:

restoreDiacritic(String)	Obnoví diakritiku v textu, použije vždy nejpravděpodobnější možnost, slova kde tato pravděpodobnost nepřesahuje nastavenou mez mohou být zvýrazněna. Inicializuje interní strukturu, jedná se o klíčovou funkci volanou při spuštění doplňování.
setListAlternatives()	Po výběru slova v textu se volá tato metoda, která nastaví seznam možných slov tak, aby uživatel mohl některé vybrat.
listAlternativesSelected()	Jakmile uživatel vybere některé slovo, zavolá se tato funkce, které vybrané slovo dosadí do textu a případně jej doplní i dále do textu pro stejná slova, je-li tato funkce povolena v nastavení.
setupUpdate()	Změní-li uživatel některé nastavení, zajistí tato metoda jeho korektní aplikaci.

6.3.7 Třída DiacriticRemover

Jednoduchá třída, jejíž funkcí je odstranění diakritiky ze zadaného textu (metoda removeDiacritic). K odstranění pomáhá tabulka, která mapuje každý znak s diakritikou na odpovídající znak bez diakritiky.

6.3.8 Třída Setup

Tato struktura udržuje hodnoty veškerého nastavení programu, konstruktor zajišťuje implicitní nastavení. Další třídy reprezentují dialogová okna.

6.3.9 DialogSetup

Okno pro veškeré nastavení programu, je vázáno na objekt třídy Setup. Obsahuje několik voleb a tlačítka OK a Zrušit. Obsluha tlačítka OK se stará o kontrolu nastavení a převod do objektu Setup.

6.3.10 DialogAbout a DialogHelp

Jednoduchá dialogová okna pro informace o programu a nápovědu.

6.3.11 DialogMain

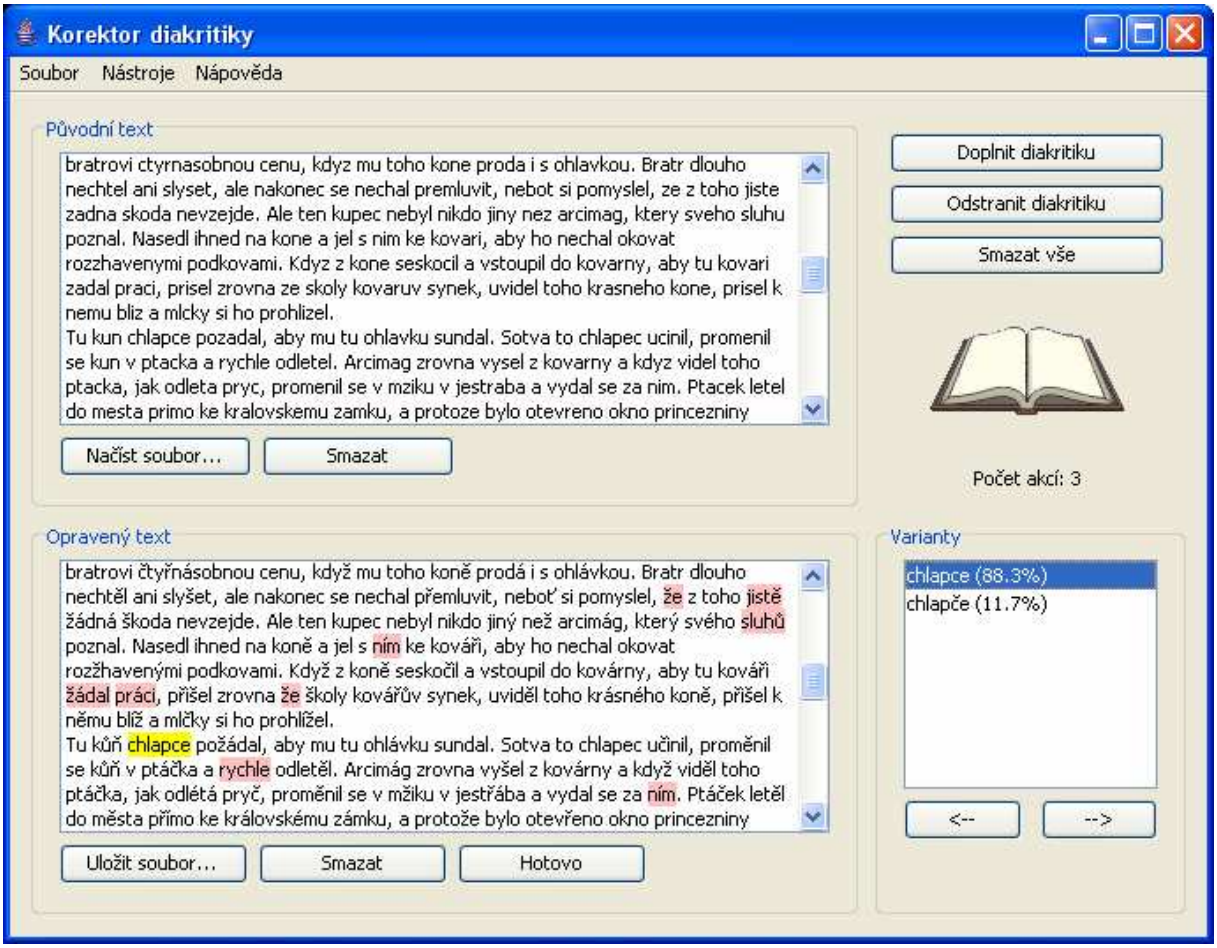
Zajišťuje vzhled samotné aplikace a obsahuje vazby na důležité objekty které zajišťují funkčnost celé aplikace (jádro). Při postupném doplňování a úpravách textu se stará o provedené změny, pohybuje kurzorem v textu a nabízí uživateli výběr s možnostmi pro přepsání. Nejdůležitější akcí je spuštění doplnění/odstranění diakritiky.
Doplnění diakritiky zajišťuje objekt třídy DiacriticRestorer. Ten projde původní text a ke každému slovu vybere tu nejpravděpodobnější alternativu s diakritikou. Není-li pravděpodobnost dostatečně velká (mez lze nastavit v nastavení), je slovo označeno červeně. Slova, která se ve slovníku vůbec nevyskytují, jsou označena šedou barvou, i toto lze v nastavení vypnout. Po tomto základním doplnění se uživatel může pohybovat po slovech (kurzorovými šipkami na klávesnici, případně myší). Výběr slova způsobí zobrazení všech alternativ v seznamu, je na uživateli, aby vybral správnou alternativu. Je-li s doplněním uživatel spokojen, stiskem tlačítka 'Hotovo' ukončí doplňování, všechna zvýraznění se zruší a text je povolen k ruční editaci.

Odstranění diakritiky je jednodušší proces, který nevyžaduje zásah uživatele. Původní text s diakritikou se vloží do horního textové pole a volbou akce 'Odstranit diakritiku' se provede akce. Odstranění zajišťuje objekt třídy DiacriticRemover, výsledek se zobrazí v dolním textovém poli.

6.4 Grafická podoba aplikace

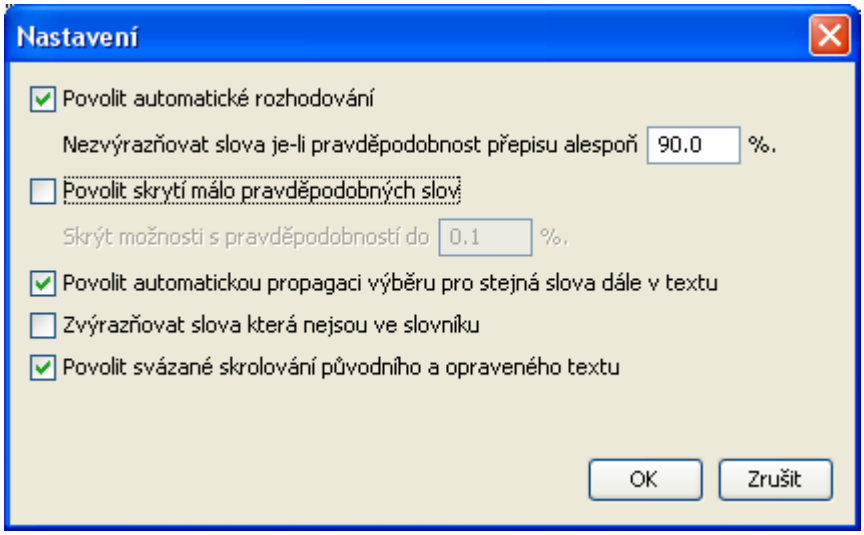
Nyní se blíže podíváme na grafickou stránku vytvářeného programu. Rozvržení jednotlivých částí bylo určováno tak, aby se kladl důraz na intuitivní ovládání a jednoduchou práci s aplikací. Výhodou může být, že celkový vzhled je automaticky přizpůsobován vzhledu operačního systému, ve kterém je program spuštěn. Na obrázku 6.1, můžeme vidět pohled na aplikaci, která je využívána v operačním systému MS Windows.

Pracovní okno je logicky rozčleněno do tří hlavních oblastí. Horní oblast se skládá z textového pole pro původní text a tlačítek pro načtení a smazání textu. Dolní oblast je velmi podobná pouze s tím rozdílem, že se týká opraveného textu. Třetí částí je pak pravý panel, který obsahuje tlačítka pro spuštění samotného doplňování a seznam variant pro aktuální slovo. Veškeré změny nastavení nebo nápovědu ohledně ovládání programu lze nalézt v horní liště.



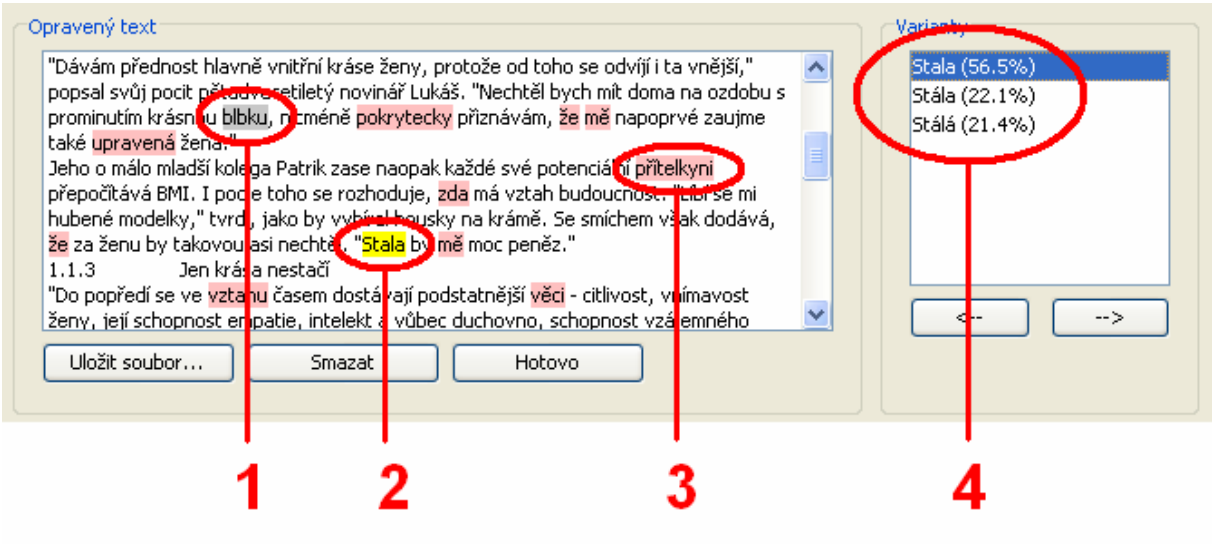
Obrázek 6.1 - Ukázka vzhledu aplikace

Pro změny v nastavení způsobu doplňování diakritiky má uživatel na výběr několik možností. Díky přizpůsobení chování programu svým požadavkům může zefektivnit, zrychlit nebo naopak zpřesnit svůj překlad. Dialogové okno pro nastavení chování aplikace můžeme vidět na obrázku 6.2. Podrobnější vysvětlení k jednotlivým možnostem nastavení byly uvedeny v kapitole 6.1.2, můžeme snad jen doplnit, že svázané rolování textu znamená, že použití posuvníku u jednoho textového okna způsobí posunutí i druhého textového okna na stejnou pozici. Při rozsáhlejším textu tato volba zpřehledňuje korekci a následnou kontrolu výsledků. Počet akcí udává, kolik stisků klávesy nebo tlačítka myši, jsme již při doplňování diakritiky použili.



Obrázek 6.2 - Možnosti nastavení aplikace

Nyní se blíže podíváme na výstup programu, pokud chceme diakritiku doplnit. Do spodního okna je vypsán celý přeložený text, který uživatel může volně prohlížet. Postupně je veden po sporných slovech, u kterých je podle nastavení vyhodnocena možnost jejich chybného přeložení. Z nabízených možných variant buď potvrdí navrhovaný překlad, nebo vybere ze seznamu ten správný. Po takto přeloženém textu je uzavřen poloautomatický překlad tlačítkem hotovo a uživatel může následně ručně editovat slova, která například nejsou zapsána ve slovníku, nebo která automatický překlad chybně přeložil.



Obrázek 6.3 - Výstup aplikace

Popis výstupu aplikace podle obrázku 6.3:

1. Přes poměrně velkou slovní zásobu nalézající se ve slovníku se v překládaném textu mohou vyskytnout výrazy, které jsou neznámé. V takovém případě jsou označeny šedou barvou a

vyžadují zvýšenou pozornost při následném ručním doplňování, protože se v nich může vyskytnout diakritika, kterou program nebyl schopen doplnit.

2. Žlutou barvou je vyznačeno právě aktuální slovo, které uživatel upravuje. Během ručního doplňování projde takto postupně v textu všechna slova, která jsou vyznačena červeně.
3. Červeně jsou označena taková slova v textu, kde je potenciální možnost chybného překladu. Tato hranice pro automatický překlad je volitelná, uživatel se pohybuje pouze po takto vyznačených slovech.
4. V pravé části pracovního okna jsou nabízeny všechny dostupné varianty pro překlad. Je u nich uvedena i pravděpodobnost, s jakou se vyskytují a podle této sestupně seřazeny. Nejpravděpodobnější přepis je tedy na prvním místě a záleží pouze na uživateli, jakou variantu si v konečné fázi vybere.

V této kapitole jsme si objasnili postupy, které byly použity při vytváření praktické části této diplomové práce. Snažili jsme se blíže seznámit a popsat konkrétní činnosti, které aplikace nabízí. Samotné praktické využití a vyhodnocení výsledků práce uživatelů s aplikací bude předmětem následující kapitoly.

7 Praktické výsledky

V předcházející kapitole jsme popsali chování a postupy při vytváření aplikace. Ukázali jsme si některé možnosti, které program nabízí, a nyní budeme sledovat, jak se jeho funkčnost projeví při reálném užívání. Zřejmě by bylo velmi málo uživatelů, kteří by chtěli používat vytvořený program, který je nepřehledný, jeho ovládání značně komplikované a výsledky, kterých dosahuje, jsou velmi špatné. Abychom se takovým negativním reakcím vyhnuli, je třeba chování celé aplikace testovat na uživatelích a reálných datech. Dosud jsme pro dokumentování úspěšnosti překladu používali pouze tzv. data testovací, která byla získána jako soubor velkého množství údajů. Tato vyhodnocení byla prováděna v teoretické oblasti a sloužila pro vhodné vytvoření slovníku dat. Zde bude kladen důraz na praktické použití a porovnání, do jaké míry se výsledné údaje shodují s původním předpokladem.

Pokud bychom chtěli výsledky této práce nějakým způsobem zhodnotit, je vhodné je srovnávat s již existujícími aplikacemi. Už pro první seznámení s problematikou doplňování diakritiky byla použita volně dostupná aplikace *czaccent*. Jako postupné cíle jsme si vytyčili na tuto práci navázat a v konečné fázi vývoje pokud možno vykazovat lepší výsledky, jak v úspěšnosti překladu, tak v rychlosti a efektivitě práce.

Program *czaccent* je postaven na unigramovém modelu používaného slovníku. Znamená to, že pro přepis na slovo s diakritikou je vždy vybrán takový výraz, který se z nabízených možností jeví jako nejpravděpodobnější. Zbylé možnosti nemusí být ve slovníku vůbec obsaženy, protože nejsou pro přepis nikdy použity. Tato varianta je tudíž méně náročná na paměť, která je využita při vytváření samotného slovníku. Jeho úspěšnost doplnění je také poměrně vysoká, přesto jsme se ji snažili ještě zpřesnit.

7.1 Přesnost doplnění

Pro praktické testování nám posloužily náhodně vybrané texty z internetu o přibližné velikosti rozsahu formátu A4. Takto získaná data byla nejprve zbavena diakritiky a následně použita v programu *czaccent* a v naší vyvíjené aplikaci. Postupně proběhlo opravování textu pro různé odchylky, které určují hranici mezi automatickým přepisem a zásahem uživatele. Výsledky testování jsou zachyceny v Tabulce 7.1. Sledováno je množství nesprávně přeložených výrazů a také míra rozhodování uživatele, která se při vzrůstající odchylce zvyšuje.

	czaccent	0%	60%	70%	80%	90%	95%	100%
Text 1 - chyby	6	5	3	1	1	1	0	0
Text 1 - rozhodování	0	0	4	5	14	24	39	403
Text 2 - chyby	20	10	5	5	5	2	1	0
Text 2 - rozhodování	0	0	14	17	34	58	75	415
Text 3 - chyby	15	12	7	4	3	1	1	0
Text 3 - rozhodování	0	0	11	26	41	57	72	433
Průměr - chyby	**13,67**	**9,00**	**5,00**	**3,33**	**3,00**	**1,33**	**0,67**	**0,00**
Průměr - rozhodování	**0,00**	**0,00**	**9,67**	**16,00**	**29,67**	**46,33**	**62,00**	**417,00**

Tabulka 7.1 - Praktické výsledky korekce

Komentář k tabulce:

chyby – počet výrazů, u kterých byla diakritika doplněna chybně a neshodovala se s původním vzorem před zbavením háčků a čárek

rozhodování – množství výrazů, které nejsou rozhodovány automaticky, ale jejich opravení obstará uživatel. U programu *cz_accent* je doplnění prováděno plně automaticky, tudíž uživatel nemá možnost učinit korekci ručně. Totéž platí i při nastavené přesnosti 0%, poloautomatické rozhodování nastává až při vzrůstající přesnosti.

procentuální přesnost – v tabulce je sledováno postupné zvyšování hranice mezi automatickým doplněním diakritiky a manuálním rozhodnutím. Pokud je tedy například hranice nastavena na 80%, rozhodují se automaticky pouze takové výrazy, kde je pravděpodobnost doplnění na určitou variantu vyšší. Nemá smysl mapovat změny, které nastanou při přesnosti 10% - 50%. Výrazů, které mají všechny možnosti přepsání s pravděpodobností menší než 50%, je obecně velmi málo a z výsledků je patrné, že se ve vyhodnocování významně neodrazí.

Při analyzování výsledků se zaměříme na porovnání aplikace *cz_accent* s naším programem. Jestliže by byl v obou případech použit pro doplňování diakritiky stejný slovník, mají být výsledky pro nulovou odchylku totožné. To se ovšem ani v jednom testovacím případě nestalo a výsledky hovoří ve prospěch naší aplikace. Znamená to tedy, že pro tento program používáme lépe vytvořený slovník, který přesněji odhaduje možné korekce a vykazuje menší chybovost. Při bližším porovnání zjišťujeme, že náš slovník obsahuje větší množství méně obvyklých jmen vlastních, příjmení nebo třeba hovorových výrazů. V takových případech program *cz_accent* chybně ponechá výraz bez doplnění diakritiky a vyhodnotí jej jako neznámý. Také se ve výjimečných případech může lišit přepis slov, kde je pravděpodobnost doplnění několika variant vyrovnaná a liší se pouze o velmi malé hodnoty.

Můžeme tedy říci, že co se týká přesnosti překladu, podařilo se nám úspěšně navázat na existující program *cz_accent*. Zkvalitnění doplnění je patrné již při nulové odchylce a dále

zpřesňováno díky zvyšování hranice pro automatický překlad. Samozřejmě to má ale za následek častější zásah uživatele do prováděné korekce.

Z tabulky je dále patrný prudký nárůst počtu výrazů, kterými se musí uživatel zabývat, při zvolení odchylky 100%. Je to dáno tím, že v tomto případě se musí rozhodnout všechna slova, která mají k sobě alternativu i se zanedbatelnou možností přepisu. Toto nastavení je značně neefektivní, velmi zdlouhavé a časově náročné, i když vykazuje konečnou stoprocentní úspěšnost doplnění.

Pokud bychom chtěli určit kvalitu překladu v procentech a zjistit relativní zpřesnění překladu při postupném navyšování odchylky, nahlédneme do Tabulky 7.2. Hodnoty z tabulky jsou vypočítány z počtu slov v jednotlivých textech, z množství chybných přepisů a počtu rozhodování uživatelem.

	cz_accent	**0%**	**60%**	**70%**	**80%**	**90%**	**95%**	**100%**
Text 1 - správný překlad (%)	98,81	99,01	99,41	99,80	99,80	99,80	100,00	100,00
Text 1 - rozhodování (%)	0,00	0,00	0,79	0,98	2,77	4,74	7,71	79,64
Text 2 - správný překlad (%)	96,87	98,44	99,22	99,22	99,22	99,69	99,84	100,00
Text 2 - rozhodování (%)	0,00	0,00	2,19	2,66	5,32	9,08	11,58	64,95
Text 3 - správný překlad (%)	97,17	97,74	98,68	99,25	99,43	99,81	99,81	100,00
Text 3 - rozhodování (%)	0,00	0,00	2,08	4,91	7,74	10,75	13,58	81,70
Průměr - správný překlad (%)	**97,62**	**98,40**	**99,10**	**99,42**	**99,48**	**99,77**	**99,88**	**100,00**
Průměr - rozhodování (%)	**0,00**	**0,00**	**1,69**	**2,85**	**5,28**	**8,19**	**10,96**	**75,43**
Relativní zlepšení (%)	**-**	**32,78**	**43,75**	**35,56**	**10,34**	**55,77**	**47,83**	**100,00**

Tabulka 7.2 – Procentuální výsledky praktického užití

Pokud srovnáme praktické použití aplikace s prvotním teoretickým testováním v páté kapitole, zjistíme, že jsme dosáhli vyšších hodnot přesnosti překladu. Je to způsobeno podrobnějším slovníkem, který pro naší aplikaci využíváme. Při teoretickém testování byl slovník postupně sestavován z části dat, která jsme měli k dispozici. Ovšem ta část dat, která se následně překládala ve slovníku, chyběla. S vyšší pravděpodobností se tedy stávalo, že výrazy nebyly ve slovníku nalezeny nebo byla hůře vyhodnocena jejich pravděpodobnost přepisu. Zatímco v naší aplikaci již používáme komplexnější soubor dat, s vyšší přesností jejich poměru četností.

Jestliže chceme stanovit optimální nastavení pravděpodobnosti přepisu pro automatické rozhodování, musíme vycházet ze zjištěných výsledků. Měli bychom zvolit takovou hodnotu, která by byla vhodným poměrem mezi počtem vyskytujících se chyb a množstvím výrazů, které se nerozhodují automaticky. Samozřejmě, že konečné rozhodnutí závisí na uživateli, jestli dá přednost stoprocentní správnosti textu nebo rychlejší práci s aplikací. Na základě dosažených výsledků bych doporučoval stanovit hranici v rozmezí 70% - 90%.

7.2 Časová náročnost

Dosud jsme se při užívání aplikace zaměřili pouze na stanovení přesnosti překladu nebo na míru poloautomatického rozhodování. Nezanedbatelnou složkou je ale také rychlost, s jakou je uživatel schopen diakritiku doplnit. Zřejmě by nebylo na místě, kdyby aplikace vykazovala stoprocentní úspěšnost, ale uživatel by s ní musel strávit daleko více času, než za který by korekci háčků a čárek provedl bez použití programu.

Již jsme dokázali, že co se týká přesnosti překladu, je naše aplikace výkonnější než *cz_accent.* Nyní zkusíme porovnat i jejich rychlosti překladu, z pohledu uživatelského. Vycházíme z předpokladu, že po provedení doplnění diakritiky bychom chtěli mít text, který neobsahuje žádný chybný přepis. Ani jedna z aplikací nevykazuje stoprocentní automatický překlad, proto musíme text po korekci celý přečíst, zkontrolovat a popřípadě opravit vzniklé nepřesnosti. Tato doba bude jistě minimálně taková, jak dlouho nám bude trvat vybraný celý text pouze přečíst. Následně se navyšuje podle toho, kolik času strávíme opravami chybného překladu.

Testování bylo prováděno experimentálně a byl měřen čas, za který jsou schopni jednotliví uživatelé bezchybně přeložit určitý text s použitím zmíněných aplikací. Srovnáváme také s časovým údajem, za který uživatel doplní diakritiku v běžném textovém editoru bez využití žádných prostředků. Zjištěné výsledky jsou znázorněny v Tabulce 7.3.

	Uživatel 1	Uživatel 2	Uživatel 3	Uživatel 4	Uživatel 5	**Průměr**
Přečtení textu (min)	2:42	2:48	3:04	2:51	3:15	**2:56**
Použití *Korektoru* (min)	3:08	3:57	3:48	4:06	4:13	**3:51**
Nárůst času (%)	16,05	41,07	23,91	43,86	29,74	**31,25**
Použití *cz_accent* (min)	3:39	4:24	4:13	4:20	4:36	**4:14**
Nárůst času (%)	35,19	57,14	37,50	52,05	41,54	**44,32**
Ruční doplnění (min)	19:47	20:05	19:17	21:15	22:04	**20:29**
Nárůst času (%)	632,72	617,26	528,80	645,61	578,97	**598,29**

Tabulka 7.3 – Srovnání časové náročnosti

Komentář k tabulce:

Všechny hodnoty časových úseků jsou uváděny s přesností na sekundy. Při použití našeho Korektoru byla pravděpodobnost pro automatické rozhodování nastavena na 70%. Tato hodnota byla vyhodnocena jako optimální pro standardní potřeby při doplňování diakritiky.

Přečtení textu – nejmenší možný čas potřebný pro kontrolu správnosti přepisu. Pokud by po doplnění text neobsahoval žádné chyby a uživatel by se nemusel rozhodovat u jednotlivých variant

překladu, doba pro překlad by byla rovna právě této hodnotě, kdy by došlo pouze ke kontrole správnosti textu.

Použití aplikací – taková doba, která je potřeba na bezchybné doplnění diakritiky, při použití jednotlivých nástrojů. Delší časový úsek proti pouhému přečtení je způsoben ručními zásahy přímo do textu a také rozhodováním uživatele z více variant přepisu.

Nárůst času – procentuální vyjádření, o jaký časový úsek je doplnění delší než samotné přečtení textu.

V porovnání se zdlouhavým a namáhavým doplňováním diakritiky v textovém editoru jsou oba sledované nástroje zhruba stejně účinné. Pokud bychom ale chtěli srovnávat pouze tyto aplikace, pak z údajů, které jsou k dispozici, je patrné, že náš program je, co se týká časového hlediska, efektivnější než *cz_accent.* Je to způsobeno rozdílným přístupem k chybným překladům v každém nástroji. V *cz_accent* uživatel nemá možnost výsledný text editovat a pro následnou práci s ním jej musí nejprve přesunout a poté může chyby v překladu opravovat ručně. Naše práce je naopak postavena na výběru ze všech dostupných variant a volné editaci textu v pracovním prostředí. Při následných opravách chyb nemusí dojít k ruční korekci, stačí, když uživatel klikne na problematické slovo a jsou mu okamžitě nabídnuty všechny možnosti změny. Takto může provést doplnění diakritiky s minimální časovou ztrátou.

7.3 Aplikace programu na dokumentaci

Na závěr této práce jsme se pokusili ověřit efektivitu vytvořené aplikace tím, že jsme doplnili diakritiku v této předkládané dokumentaci. Nejprve jsme diakritiku odstranili a výsledný text použili jako vstup pro náš Korektor. Sledovali jsme míru chybovosti a vynaloženou námahu uživatele.

Výsledku testu odpovídají hodnotám, které jsme získali na náhodně vybraných textech. V textu dokumentace se vyskytovalo celkem 14015 slov. Při nastavení optimální hranice pro automatický překlad 70%, činila míra rozhodování 4,72%. Úspěšnost správného doplnění byla vyčíslena na 98,53%. Chybné přepisy vznikaly především u odborných výrazů, které se opakovaly ve větší míře nebo u slov jako například *ze,* které se chybně opravovalo opakovaně na *že.* Pro kompletní bezchybné doplnění diakritiky v dokumentaci bylo potřeba 1265 stisků klávesy nebo myši.

Ukázali jsme si tedy, že se nám podařilo úspěšně navázat na vytvořenou aplikaci *cz_accent* a v mnoha ohledech dokonce překonat. Výsledky ukazují, že náš Korektor diakritiky má menší procento chybných překladů a jeho časová náročnost při praktickém použití je nižší. Práci s tímto programem navíc ulehčuje množství nastavení, která přizpůsobí jeho chování každému uživateli. Ověřili jsme si tedy, že je námi vytvořená aplikace schopná doplňovat diakritiku ve velmi krátkém čase i při minimální chybovosti překladu.

8 Závěr

Cílem předložené práce byla implementace aplikace na doplňování a odstraňování diakritiky v textu. V úvodní kapitole je tato problematika představena a nastíněny hlavní problémy, které jsou s diakritikou spojeny. Jelikož je v naší aplikaci použita rozsáhlý slovní zásoba, museli jsme se zamyslet nad jejím vhodným uložením. Ukázali jsme si, že nejvhodnější je pro tento účel datová struktura trie a také jsme využili její analogii s konečným automatem pro její minimalizaci. Popsali jsme některé metody, které jsou využívány při doplnění diakritiky a určili jsme, která je pro naše potřeby vhodná.

Abychom mohli co nejpřesněji vyhodnocovat možnosti při korekci textu, bylo nutné vytvořit odpovídající slovník. Nejprve byla analyzována a zpřesňována úspěšnost překladu na testovacích datech a ujasňován vhodný tvar a rozsáhlost slovníku. Následně byla nad vzniklým slovníkem vytvářena samotná aplikace, která byla v konečné fázi podrobena reálnému testování náhodně vybraných uživatelů. Původně měla aplikace navazovat na program *cz_accent*, založený na unigramovém modelu při výběru variant pro překlad. Praktické testování prokázalo, že náš Korektor diakritiky vykazuje vyšší úspěšnost překladu a také nižší časovou náročnost než výše zmíněný nástroj.

Přínosem programu je jistě urychlení práce, která je spojena s namáhavým a zdlouhavým přepisováním textu v situaci, kdy je to nejméně potřeba. V dnešní době se domníváme, že požadavky na podobné nástroje pro urychlování činnosti ve všech možných sférách budou nadále přibývat.

Můj cíl sestavit jednoduchou efektivní aplikaci na interaktivní doplňování diakritiky se podařilo splnit. V rámci řešení projektu jsem se seznámil s technologií Java, naučil jsem se základům návrhu grafického uživatelského prostředí a v neposlední řadě si v praxi procvičil objektově orientovaný návrh a implementaci. Těchto zkušeností si velmi cením.

Úspěšnost a efektivitu programu je možné nadále vylepšovat. Ideálním výsledkem je dosažení stoprocentně správného doplnění, které by mohlo být realizováno použitím nástrojů umělé inteligence a také například segmentací a morfologickou a syntaktickou analýzou slov.

9 Literatura

[DAC1-98] Daciuk Jan: Finite state utilities, 1998. Nástroje dostupné na URL: http://www.eti.pg.gda.pl/katedry/kiw/pracownicy/Jan.Daciuk/personal/fsa.html (květen 2007)

[DAC2-98] Daciuk Jan: Incremental construction of finite-state automata and transducers, and their use in the natural language processing, 1998, Politechnika Gdańska

[DAC3-02] Daciuk Jan, van Noord Gertjan: Finite automata for compact representation of language models in NLP, 2002. Dokument dostupný na URL: http://www.springerlink.com/content/917gy4balttqguet/fulltext.pdf (květen 2007)

[JAN-04] Janovský Dušan: Češi hledají s diakritikou, 2004. Dokument dostupný na URL: http://www.jakpsatweb.cz/clanky/hledame-s-diakritikou.html (květen 2007)

[KON-06] Wikipedie, encyklopedie, Konečný automat. Dokument dostupný na URL : http://cs.wikipedia.org/wiki/Kone%C4%8Dn%C3%BD_automat (květen 2007)

[MIH-02] Mihalcea Rada F.: Diacritic restoration, 2002. Dokument dostupný na URL: http://www.springerlink.com/content/wxpjmp1y8hdk8rvv/fulltext.pdf (květen 2007)

[NLP-07] Laboratoř zpracování přirozeného jazyka. Materiály dostupné na URL: http://nlp.fi.muni.cz/nlp/aisa/NlpCz/LaboratorNLP.html (květen 2007)

[SED-99] Sedláček, Radek, Morfologický analyzátor češtiny, 1998, Masarykova univerzita Brno, Fakulta informatiky.